Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

TESIS DE LICENCIATURA

IGNATIUS
Una Herramienta para Construir Sistemas de Supervisión en Tiempo Real

Silvia V. Benítez, Juan J. Seoane,
Gabriel A. Wainer, Roberto J.G. Bevilacqua.

Pabellón 1 - Planta Baja - Ciudad Universitaria
(1428) Buenos Aires
Argentina

http://www.dc.uba.ar

TABLA DE CONTENIDOS

IGNATIUS[1]
Herramienta para el Desarrollo de Aplicaciones de Tiempo Real

Silvia V. Benítez - Juan J. Seoane
Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

ABSTRACT

In the present, there is a large number of computers in the industry controlling processes of the real world and running Real Time applications. The feedback loop control techniques have not changed so much from the beginning of the use of computers for process control; in the role of *process supervision* is where most of the changes have ocurred. Although, the applications developed for this purpose are not flexible enough and have a very limited range of use. There are a few operating systems and programming languages that provide facilities for the execution of tasks with hard time constrains. Also, there are not too much development environments that solve all the problems related with the restrictions of this type of systems, therefore, the applications continue being built "ad-hoc" for each of the problems.

This paper addresses our work of design and implementation of a *Tool to Build Real-Time Supervisory Systems: IGNATIUS.*

An architectural decomposition of all the Supervisory Systems in three service levels has been proposed: Interface Service Level, Particular Service Level and Basic Service Level. The ISL and PSL vary with each particular implementation, but the BSL is common for all of them. IGNATIUS encapsulates the BSL. This decomposition limits the effort of building supervisory systems to the work of building the ISL and PSL, shortenning dramatically the development cicle, reducing the complexity and letting the user concentrate on high level design aspects without taking care about time constrains imposed by this type of systems.

IGNATIUS provides the user with a complete development environment that can be easily used by unexperienced programmers, helping them to maintain the integrity of the system, make changes and develop correct applications.

With this approach in mind, and considering that the BSL encapsulates the major portion of any supervisory system, a great part of the work of developing supervisory systems has been solved.

[1] IGNATIUS es un nombre de fantasía, no representa a ninguna sigla

1. INFORME CIENTÍFICO

1.1 Introducción

En nuestros días el número de sistemas de computadoras que se usan para controlar procesos del mundo real ha crecido notablemente. Este crecimiento se debe en gran medida a los rápidos avances en el diseño de hardware y a la reducción de los costos; esto está demostrado claramente por el uso del microprocesador como un componente común en una amplia gama de aparatos electrónicos. Esto ha permitido automatizar procesos del mundo real en los cuales se realizaban tareas manuales que necesitaban una respuesta inmediata, alcanzando un control automático del entorno en tiempo real. Por ejemplo, permitió operar automáticamente una fábrica de acero y optimizar su eficiencia; monitorear un vuelo espacial; controlar procesos químicos; brindar a los clientes de un banco detalles de sus cuentas al momento; planificar el trabajo de una fábrica y replanificarlo cada vez que se presentan nuevos requerimientos, o cuando la situación de la planta cambia; permite controlar el flujo de tránsito en una ciudad, detectando las posiciones de los vehículos y cambiando las luces de los semáforos de la manera óptima.

Todos estos son ejemplos de procesos que necesitan una respuesta en tiempo real.

Un sistema en tiempo real es aquel en el que la correctitud de los cálculos no solo depende de la correctitud lógica del cálculo, sino también del momento en el que se produce el resultado. Si las restricciones de tiempo no se cumplen, se dice que ha ocurrido una falla en el sistema. Luego es esencial que se garantice el cumplimiento de las restricciones de tiempo.

Un gran número de aplicaciones en tiempo real se encuentran en la industria, y en la actualidad existe un número creciente de computadoras controlando procesos industriales. Esto ha aumentado el número de actividades que se automatizaron. No sólo la computadora puede directamente controlar la operación de la planta, sino que además puede proveer a los gerentes e ingenieros de una fotografía del estado de su funcionamiento. Este rol se conoce como Supervisión de Procesos. Es aquí y en la presentación de la información al operador de la planta, donde han ocurrido la mayoría de los cambios ya que las técnicas usadas en la mayoría de los casos de control retroalimentado de la planta han cambiado muy poco desde los días en los que la norma eran los controladores operados neumáticamente.

A pesar de ello, las principales aplicaciones de supervisión tienen un rango de utilización muy limitado y son muy poco flexibles. Además, existen pocos Sistemas Operativos y muy pocos lenguajes de programación que provean facilidades relacionadas con la ejecución de tareas con restricciones duras de tiempo.

No existen aún demasiados entornos de desarrollo que solucionen todos los problemas relacionados con las restricciones en este tipo de sistemas y por ende, las aplicaciones siguen construyéndose "ad hoc" para cada uno de los problemas.

Con este panorama en vista, la propuesta de esta Tesis es el diseño e implantación de una herramienta que permita el desarrollo de un grupo importante de aplicaciones de Tiempo Real (las de Supervisión de Procesos), facilitando el trabajo de los diseñadores.

Con este fin, se hará un amplio estudio de las principales técnicas existentes en la actualidad, y se adaptará la herramienta a la técnica elegida. Con este mismo fin se hará un estudio cuidadoso del Software de Base a elegir para obtener un entorno de desarrollo completo que pueda ser utilizado por

programadores no experimentados, favoreciendo la seguridad, mantenibilidad y correctitud de las aplicaciones desarrolladas.

Esta herramienta estará diseñada como un conjunto de objetos independientes, para permitir la construcción de variedad de aplicaciones de supervisión con distintas finalidades con muy poco esfuerzo. Además se implementará un Ejecutivo de Tiempo Real, ejecutando sobre el entorno de desarrollo seleccionado. Este realizará funciones de planificación de alto nivel, permitiendo que los programadores puedan concentrarse en la resolución de las aplicaciones.

Un objetivo primordial de este trabajo es su utilización final en un curso semestral a ser dictado en nuestro Departamento, y su integración con otros trabajos de licenciatura que puedan utilizarlo como sistema de información base para aplicaciones de Tiempo Real (en particular, se tratará de integrarlo con diversos trabajos de investigación y desarrollo en el marco del proyecto EX-103, "Concurrencia y Sistemas Operativos").

Este trabajo está organizado de la siguiente manera: en los primeros capítulos se describen las funciones de control y supervisión de procesos. En el capítulo de Control de Procesos se analiza qué es control y cuáles son los tipos de control mas utilizados. En el capítulo de Supervisión de Procesos analizaremos en detalle las funciones de un supervisor, como está organizado y veremos una categorización de los mismos. También algunos supervisores existentes en el mercado.

Posteriormente se analizará MASCOT, una metodología para diseño de sistemas en tiempo real, y se verá qué Software de Base se utilizó en la construcción de la herramienta y el porqué de la elección.

Por último se describe cómo está compuesta la herramienta, cómo se usa, y se muestra un ejemplo de cómo construir un supervisor utilizándola.

En el último capítulo se describen las conclusiones a las que se llegaron y se plantean posibles trabajos futuros.

1.1.1 Definición de Sistemas en Tiempo Real

Un sistema en tiempo real es aquel que en lugar de realizar un proceso cuyo resultado será analizado y usado posteriormente, realiza el control automático del entorno con un tiempo de respuesta restringido. Por ejemplo el sistema que controla una planta donde se industrializa acero y hace óptima su eficiencia; un cajero automático; el sistema que hace el planeamiento de trabajo de una fábrica, lo replantea cuando hay nuevos requerimientos ó cambia la situación dentro de la fábrica; un sistema que controle el flujo de tránsito en una ciudad, detectando la posición de los vehículos y cambiando las luces de los semáforos de una manera óptima.

Un sistema en tiempo real puede definirse como el que controla un cierto entorno, recibiendo datos, procesándolos y retornando los resultados con una restricción de tiempo y que le permita afectar el funcionamiento del medio que controla, en ese momento.

Utilizaremos el término "Tiempo Real" para sistemas en los cuales :

(1) El orden de procesamiento está determinado por el paso del tiempo ó por eventos externos al sistema. Estos eventos externos no pueden predecirse a partir del estado actual del programa
(2) El resultado de un cálculo en particular puede depender del valor de la variable "tiempo" en el momento de realizar dicho cálculo, ó del tiempo que lleva realizar el proceso [Ben93].

Podemos dividir los sistemas de Tiempo Real en dos tipos :

(A) El sistema tendrá una media de ejecución, medida en un intervalo de tiempo determinado, que deberá ser menor que un "máximo" especificado para ese sistema. Estos sistemas, también se denominan sistemas de Tiempo Real "blandos".
(B)El proceso deberá ser completado dentro de un máximo de tiempo específico, todas y cada una de las veces que se ejecute. Estos sistemas, también se denominan sistemas de Tiempo Real duros" [Ben93].

En la segunda categoría se encuentran sistemas con restricciones de performance mucho mas severas que en la primera.

Un ejemplo de sistemas en tiempo real blandos son los cajeros automáticos. Es un sistema en tiempo real, ya que su ejecución está determinada por eventos externos: la transacción se inicia al colocar la tarjeta en la máquina; el proceso realizado depende de la variable tiempo: usualmente el usuario está limitado a un máximo de transacciones por día o por semana; y por último la restricción de tiempo sobre el sistema está basada en un tiempo de respuesta medio no en una restricción dura de tiempo sobre cada transacción realizada (compare el tiempo de respuesta obtenido entre las 12:00 y las 14:00 de un día viernes, y la obtenida un domingo a las 10:00 de la mañana) [Ben93].

Un ejemplo de sistemas en tiempo real duros, es el control de la temperatura en un horno para secado de material. Los componentes son ubicados en una cinta transportadora, la cual los transporta a través del horno. El horno es calentado por tres mecheros de gas, distribuidos a intervalos regulares. La temperatura en cada una de las áreas del horno es monitoreada y controlada por el operador a través de una consola.

Las restricciones de tiempo de respuesta especificadas para el buen funcionamiento del horno son duras, ya que no cumplirlas en el lapso especificado implica que el material sea expuesto a temperaturas muy elevadas por demasiado tiempo y esto podría dañar el material. Por otra parte si una de las áreas alcanza una temperatura muy elevada puede provocar el daño permanente del horno.

Muchos sistemas en tiempo real contienen partes que no es necesario que operen en tiempo real. Es importante tener esto en cuenta ya que puede simplifica el diseño e implantación del software en tiempo real.

Típicamente un sistema en Tiempo Real consta de un sistema de control y un sistema controlado. Por ejemplo, en una fábrica automatizada, el sistema controlado es el piso de la fábrica, mientras que el sistema de control está formado por computadoras, procesos y las interfaces con las computadoras y humanas que administran y coordinan las actividades del piso. Luego el sistema controlado puede verse como el entorno en el cual interactúa la computadora.

En un sistema en Tiempo Real podemos distinguir las siguientes funciones :

- *Monitoreo*: Hay que obtener información acerca del estado actual de mundo real, por ejemplo información de los instrumentos de medición (temperatura, velocidad, humedad, etc.)

- *Control y Cálculo*: Deben controlarse los valores leídos, es decir verificar por ejemplo que no pasen ciertos valores determinados como normales, o que no se de cierta combinación de ellos que es considerada como peligrosa (por ej. alta temperatura y alta presión). Estas tareas se realizan utilizando una serie de dispositivos de interfaz como por ejemplo conversores AD/DA, líneas de entrada/salida digital, generadores de pulso etc.

- *Actuación:* Se debe poder alterar el mundo real. Por ejemplo mantener la temperatura a un determinado valor.

En muchas aplicaciones, la comunicación con el operador es mas compleja que un simple panel con indicadores. Ingenieros de planta, gerente de planta, pilotos, controladores de tráfico aéreo,

operadores de máquinas, etc., pueden necesitar información detallada de todos los aspectos de la operación de la planta, aeronave, sistema de radar, etc., que se encuentren monitoreando. Estos sistemas de información suelen conocerse como Sistemas Supervisores.
En los siguientes capítulos describiremos mas a fondo los conceptos de Control y Supervisión de procesos.

1.2 Control de procesos

El Control de procesos puede definirse como "la aplicación de una acción planificada, para que aquello que se considera como objeto, satisfaga ciertos objetivos" [Miy93]. Control implica generar una señal de salida en respuesta a la entrada de datos. Un controlador acepta la información desde los sensores a intervalos regulares (o es manejado por eventos), procesa información y envía los resultados al objeto controlado a través de actuadores. La información de salida influye en el objeto controlado.
En el siglo XVIII, Watt inicia la revolución industrial con la invención del regulador de vapor, esto permite controlar la fuerza del vapor y con esta fuerza mover telares. De esta forma se inicia la automatización de fábricas de tejidos, comienzo de toda automatización y de la teoría de control.

En la década de 1940, los sistemas de control poseían una estructura mas simple que la usada en la actualidad: un operador interactuaba en forma directa con un dispositivo de control que servía para controlar un determinado objeto.

A partir de la década del '50 aparece el concepto de monitoreo y control remoto, con lo cual cambia la estructura de los sistemas de control: aparecen funciones de monitoreo y actuación, diferenciadas de las operaciones de control. También a fines de esta década se desarrollaron las primeras computadoras digitales para control industrial.

Analicemos a modo de ejemplo, como sería la operación de la planta presentada en el ejemplo del horno de secado, realizado por una computadora. La operación de esta planta requiere *monitoreo, control* y *actuación*. En la Figura 1., vemos un esquema general del sistema que opera la planta.

Las tareas de monitoreo, obtienen la información sobre el estado actual de la planta. Por ejemplo en el Horno de Secado se monitorea la temperatura de cada una de la áreas cada dos segundos. Para esto existen termocuplas ubicadas en cada una de las áreas del horno. También se monitorea la velocidad de la cinta para verificar que esta no caiga por debajo de los 250 cm/min. en menos de 30 segundos. Esto se hace a través de un sistema de pulso continuo.

Las tareas de control procesan los datos obtenidos de la planta y también determinan anomalías garantizando el correcto funcionamiento de la planta. Por ejemplo si la velocidad de la cinta cae por debajo de los 250 cm/min. en menos de 30 segundos se emite una señal de alarma; o si la temperatura en alguna de las áreas del horno varía en mas del 5% del valor de referencia del área, entonces también se emite una señal de alarma.

Las tareas de actuación, realizan la acción requerida para normalizar el funcionamiento de la planta, por ejemplo mandar señales al horno para subir la temperatura, normalizar la velocidad de la cinta transportadora o bajar la temperatura de alguna de las áreas.

Las tareas de monitoreo y actuación requieren de dispositivos de interfaz con la planta, como conversores analógicos/digitales y digitales/analógicos, generadores de pulsos, y líneas digitales de entrada y de salida.

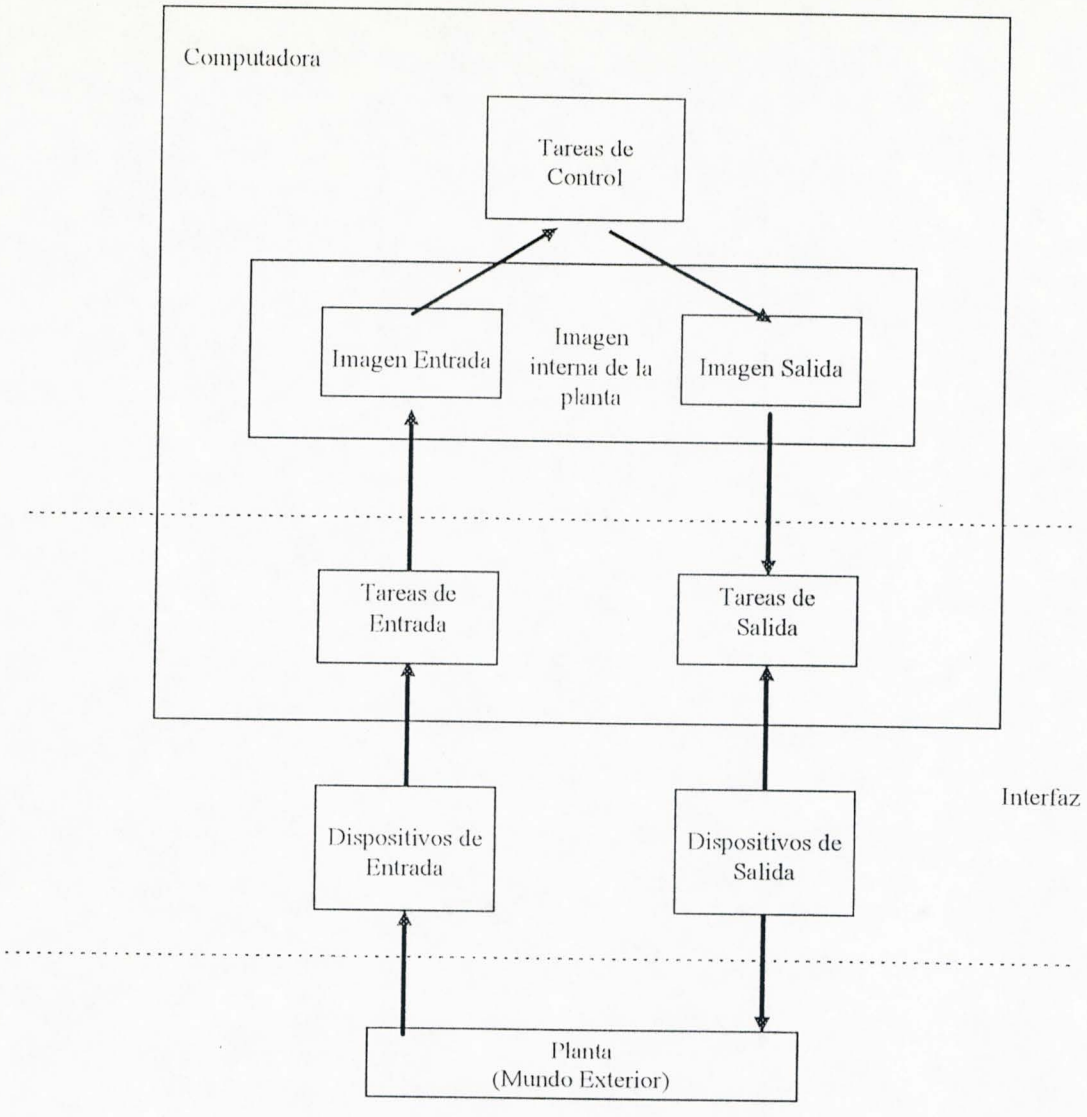


Figura 1.
Sistema de control generalizado, mostrando las interfaces de software y hardware con la planta

Por ahora los representaremos simplemente como dispositivos de entrada y de salida, como se ve en la Figura 1. Cada uno de los diferentes dispositivos, se operan mediante software especial; representaremos este software, como tareas de entrada/salida.
Los dispositivos de entrada, mas el software de entrada (tareas de entrada), proveen la información para generar la "imagen de la planta". La imagen, es como una fotografía del estado de la planta en un determinado instante. Esta imagen es renovada a intervalos previamente especificados.

Luego que las tareas de control procesan los datos de la imagen de entrada, se genera un resultado que actuará sobre la planta, esto es la imagen de salida. La imagen de salida será actualizada periódicamente por las tareas de control. Las tareas de salida trasladarán los datos encontrados en la imagen de salida, a la planta, a través de los dispositivos de salida. Podemos decir que las tareas de control actúan sobre una imagen interna (modelo) de la planta.

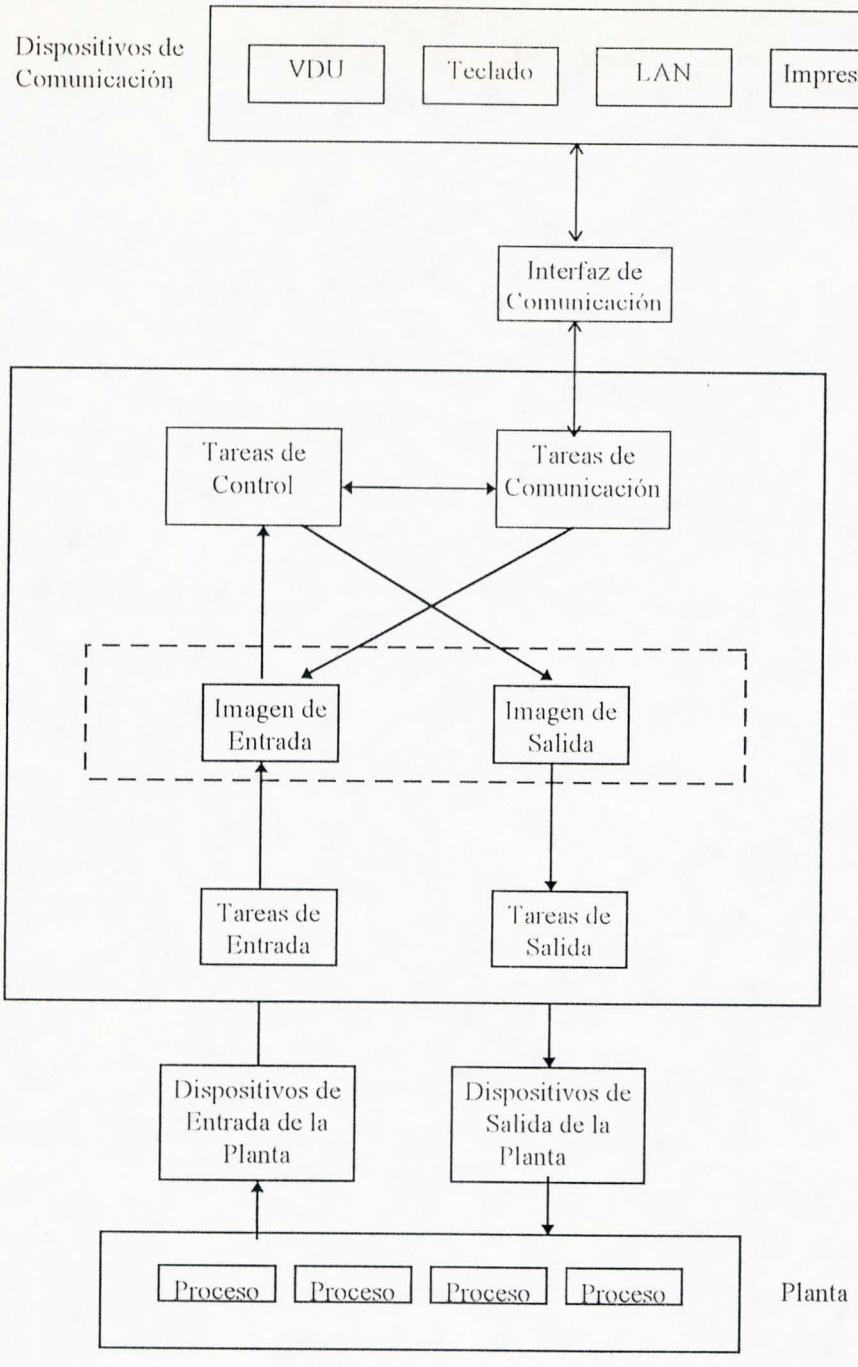


Figura 2
Sistema de Control - Tareas de Comunicación

La aparición de microprocesadores y mini computadoras usadas como controladores ha dado un nuevo impulso a esta área. Los motivos de la búsqueda de nuevos métodos de control son la productividad, la velocidad, la precisión, dependencia y el costo. Todos estos son factores íntimamente relacionados con la productividad.

También la aparición de los microprocesadores, inició el desarrollo de sistemas distribuidos, esto provocó que algunos sistemas de control tengan sus funciones distribuidas en distintas computadoras, dónde la información deberá ser transmitida entre las mismas. Podemos extender el modelo

presentado en la Figura 1., para incluir tareas de comunicación entre los distintos puestos de trabajo. Esto lo vemos en la Figura 2. La operación de la planta puede ser secuencial o en paralelo, es decir varias tareas ejecutándose concurrentemente. En este último caso deberán tenerse en cuenta la sincronización de las distintas tareas involucradas en el proceso.

1.2.1 Control Clásico

El objetivo de todo sistema de control es el modelado de un sistema físico y su posterior control automático [Wai94]. Los sistemas físicos pueden en general ser descriptos por ecuaciones diferenciales o sistemas de ecuaciones integro-diferenciales. El comportamiento de un sistema físico en un momento dado puede ser obtenido resolviéndose las ecuaciones diferenciales que lo representan, para las condiciones de entrada especificadas.

La solución de ecuaciones diferenciales puede simplificarse bastante utilizando la transformada de Laplace. Si bien no siempre es posible utilizar esta salida (ya que para que la transformada exista, es necesario que la integral de la ecuación converja), cuando se puede las ecuaciones diferenciales se resuelven fácilmente, ya que con este formalismo pueden transformarse las ecuaciones diferenciales en ecuaciones que pueden ser tratadas algebraicamente. De esta manera para resolver una ecuación diferencial, deberán seguirse los siguientes pasos :

1. Transformar la ecuación diferencial por Laplace
2. Resolver la ecuación algebraica resultante
3. Transformar la ecuación resultante a través de Laplace en una ecuación Diferencial. La ecuación obtenida de esta forma es la solución para la ecuación diferencial original.

Como dijimos al comienzo, un tema crucial es como modelar un sistema físico para poder analizar su comportamiento. Los diagramas de bloques permiten la representación de las funciones de un sistema y del flujo de señales. Esta herramienta es utilizada para modelar sistemas físicos. La representación de los componentes básicos de un diagrama de bloques pueden verse en la Figura 3 [Miy93].

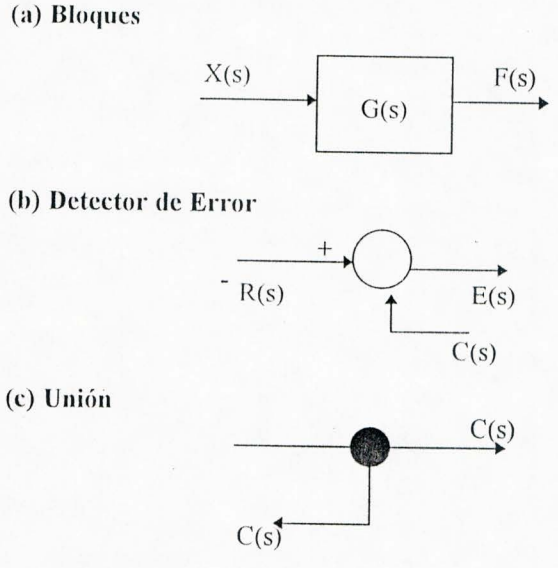


Figura 3 - Elementos de un diagrama de bloques.
(a)Bloque; (b)Detector de error; (c)Unión;

La función de un bloque llamada (función de transferencia), relaciona la transformada de Laplace de con las variables de entrada y salida. O sea: R(s) / F(s) = G(s). Con la composición de bloques, detectores de error y unión, pueden esquematizarse sistemas complejos. En los sistemas físicos es de suma importancia la respuesta temporal de los mismos. Esta respuesta está formada por la respuesta transitoria y la respuesta estacionaria. La respuesta transitoria es el comportamiento en el momento de salida de un período de tiempo dado (desde el estado inicial, hasta el estado final). La respuesta estacionaria refleja el comportamiento del sistema cuando el tiempo tiende a infinito.

Otro aspecto que debemos considerar en lo sistemas físicos es la estabilidad de los mismos. Entre todos los diversos sentidos que en teoría de control se define a la "estabilidad", nos interesa sobre todo la denominada estabilidad absoluta, que podemos traducirla como: si la entrada a un sistema físico es finita la salida debe serlo también. Un ejemplo de inestabilidad puede darse, por ejemplo, en el caso de un sistema de control de lazo cerrado de realimentación positiva. En este caso el detector de error suma el valor de entrada al valor de salida en lugar de restarlos. De esta manera no se corregirá el valor de salida sino que por el contrario aumentará su valor, lo que provocará que el sistema se vaya fuera de control.

A continuación haremos una breve referencia a los tipos de controladores mas usados en aplicaciones industriales :

Controlador ON-OFF

Este tipo de controlador es el mas simple y es muy usado en aplicaciones de control de nivel. El problema que presenta es que si por ejemplo se produce una variación brusca en el nivel de entrada del sistema, la respuesta temporal obtenida es oscilatoria.

Este tipo de controlados tendrá una salida s(t) = M1 si el error e(t) > 0; o una salida s(t) = M2 si el error e(t) <= 0.

Controlador Proporcional

Este tipo de controlador es muy usado para controlar variaciones bruscas en el nivel de entrada del sistema. Este controlador diminuye el la amplitud de oscilación de la salida. Si como en el caso anterior suponemos un cambio brusco en la entrada del sistema, se obtendrá una respuesta temporal exponencial. Una variación en la posición de entrada P, es proporcional al error e, de forma que P = Kp*e, dónde Kp es una constante de proporcionalidad.

Usando este tipo de controlador, obtendremos un desajuste entre el valor final obtenido y el valor deseado. Este desajuste puede hacerse mínimo con la elección adecuada de la constante de proporcionalidad Kp, pero nunca puede ser anulado.

Controlador Integral

Esta acción de control corresponde a un término de la forma 1/s, que permite eliminar el error anterior, ya que su respuesta estacionaria tiene un error no nulo, que combinado con el desajuste obtenido usando acción proporcional nos permite obtener una respuesta estacionaria correcta

La acción de control integral hace que en todo momento, la señal de corrección de la salida del controlador sea igual a la integral del error. Por otro lado, aunque elimina el error en la respuesta estacionaria, puede resultar en una respuesta oscilatoria

Controlador Derivativo

Este tipo de control aumenta la sensibilidad del controlador, ya que la señal de corrección de hace proporcional a la derivada del error. Esto puede provocar alteraciones significativas antes de que el valor del error sea muy alto. Por lo tanto aumentará la estabilidad del sistema. Debido a la

sensibilidad que introduce en el sistema, nunca se usa solo, sino junto con un controlador proporcional, o uno
proporcional/integral

Controlador PID

La acción de control combinada de los controladores proporcional, integral y derivativo, se denomina PID. Este tipo de controlador encuentra gran aplicación en la industria, ya que incorpora las tres acciones de control, aunque no es necesario que una aplicación dada las use todas simultáneamente.

1.2.2 Control digital

En la década del '60, con la aparición las mini computadoras, se inició el proceso de sustitución de los antiguos controladores (eléctricos, neumáticos o hidráulicos) por los controladores digitales. Si bien éstos tienen gran cantidad de ventajas (como por ej.: realizan los ajustes mecánicos necesarios, poseen dimensiones reducidas, son fáciles de operar, implementar y mantener, entre otras), la utilización del control digital se dio en forma lenta. Esto se debió en gran parte a que los precios no eran accesibles, los técnicos de las plantas e ingenieros no estaban entrenados en la programación de computadoras (sin embargo si lo estaban en RLL, relay leddear language, un lenguaje de programación basado en el diseño de circuitos con relés, usado para la programación de PLCs) y a que en esa época las mini computadoras no eran del todo confiables. Este hecho comenzó a cambiar con la aparición de los microprocesadores. Los controladores de un único lazo usados en forma autónoma, ahora se basan en técnicas digitales y contienen uno o dos microprocesadores, y son utilizados para implementar algoritmos de control digital. Muchos de ellos han sido diseñados de manera tal que pueden reemplazar a las unidades de control analógicas. Si el sistema de control digital interactúa directamente con el medio, el sistema se conoce como un Sistema de Control Digital Directo (DDC - Direct Digital Control).

1.2.2.1 Conversores AD/DA

En un sistema de control digital es necesario transformar la información analógica proveniente del mundo real, en datos discretos, es decir es necesario digitalizar la información para que la computadora pueda procesarla. Para esto existen conversores Analógicos/Digitales (AD). De forma análoga es necesario contar con conversores Digitales/Analógicos (DA), para que la computadora pueda comunicarse con el mundo real y poder así llevar a cabo alguna acción.

La conversión de mediciones analógicas a digitales involucran dos operaciones: muestreo y cuantificación. Los conversores analógicos/digitales, incluyen en el dispositivo de entrada, una unidad que permite almacenar temporariamente la muestra. Esta unidad se usa para prevenir un cambio en la muestra mientras esta está siendo convertida a una cantidad discreta [Ben93]. En la Figura 4, vemos un esquema general de un conversor AD/DA.

El método de operación de estas interfaces de entrada analógica, es que la computadora envíe una señal de 'Comienzo' o 'Muestreo', en respuesta a la cual el conversor AD, comienza a almacenar la muestra. Este proceso es muy breve y luego comienza el proceso de cuantificación. Cuando termina la conversión AD, el conversor manda una señal de 'Listo' por una de las líneas. La computadora lee esta línea, utilizando técnicas de polling o interrupciones.

Por lo general se utiliza el mismo conversor AD para las distintas entradas analógicas, por ser esto mucho mas conveniente desde el punto de vista económico. Para ello se utilizan multiplexores que permiten conectar varias líneas de entrada a un único conversor AD. En un sistema multiplexado, la secuencia de operaciones es más compleja que con un dispositivo de canal simple, ya que la

computadora debe determinar cual es el canal del que debe leer. La forma mas simple de hacerlo, es usar la señal de 'Seleccionar', ante lo cual el multiplexor pasa al próximo canal. Esto significa que los canales son muestreados en orden.

Una técnica más elaborada es proveer a los canales de entrada de direcciones de entrada, las cuales se conectan al bus de datos de la computadora. La secuencia de los eventos sería la siguiente: seleccionar el canal, y luego, lanzar el comando de conversión, y luego esperar hasta que llegue la señal de fin de la conversión. En algunos conversores de alta velocidad es posible mandar la dirección del próximo canal, mientras que los datos de entrada actuales están siendo cuantificados.

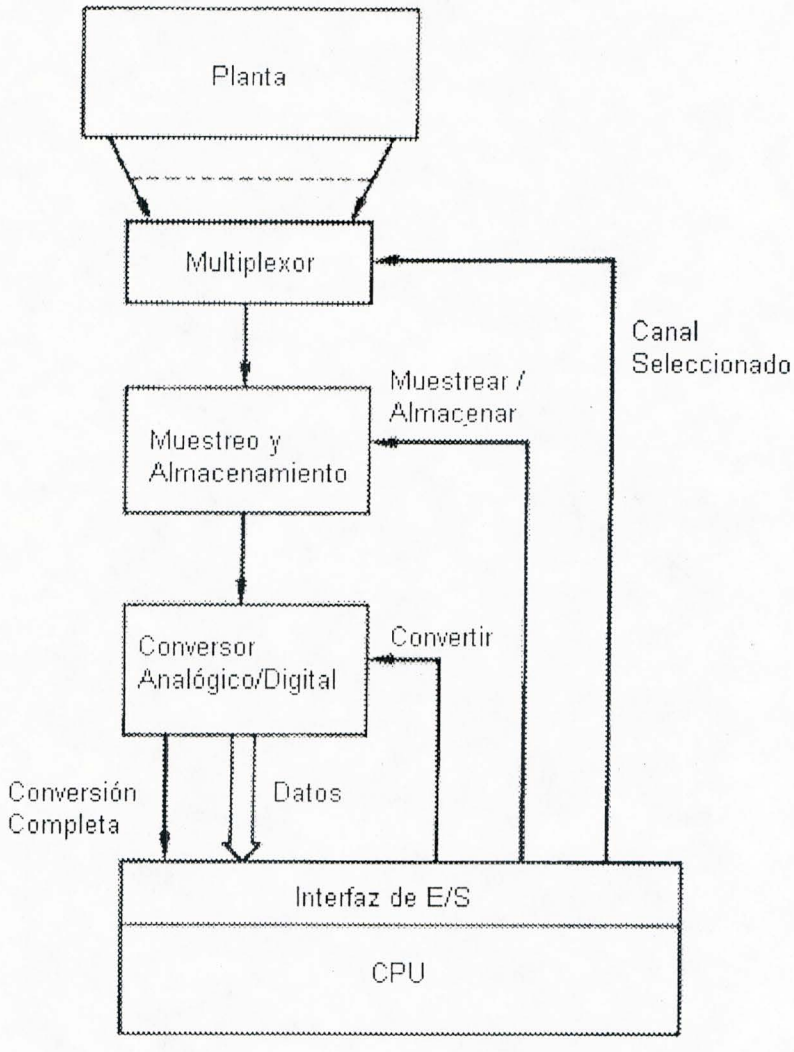


Figura 4
Conversor Analógico/Digital

La conversión Digital/Analógica es mas simple y mas barata que la conversión AD. Por este motivo es usual que haya un conversor DA por cada dispositivo de la planta que se desea controlar. Cada conversor DA, estará conectado al bus de datos, y el canal apropiado es seleccionado poniendo la dirección del mismo en el bus de dirección. El conversor DA, guarda el valor previo que se envió, hasta que llegue el próximo valor. En la Figura 5 vemos un esquema general de un conversor DA.

Como ya hemos dicho una computadora usada para el control digital de procesos, mantiene una imagen del mundo real que desea controlar. Este modelo es actualizado con datos capturados por sensores. Estos datos son transformados y procesados por la computadora. Toma un estado correcto y produce un conjunto de órdenes para lograr alguna acción en el proceso que controla. En términos formales esto se conoce como sistema de control retroalimentado: el software recibe datos de los sensores, actualiza el estado en el modelo que posee y emite órdenes a los actuadores. Los resultados de estas órdenes condicionan los valores posteriores de los datos que llegarán desde los sensores, es decir igualan el valor de una variable física a un valor de referencia [Wai94].

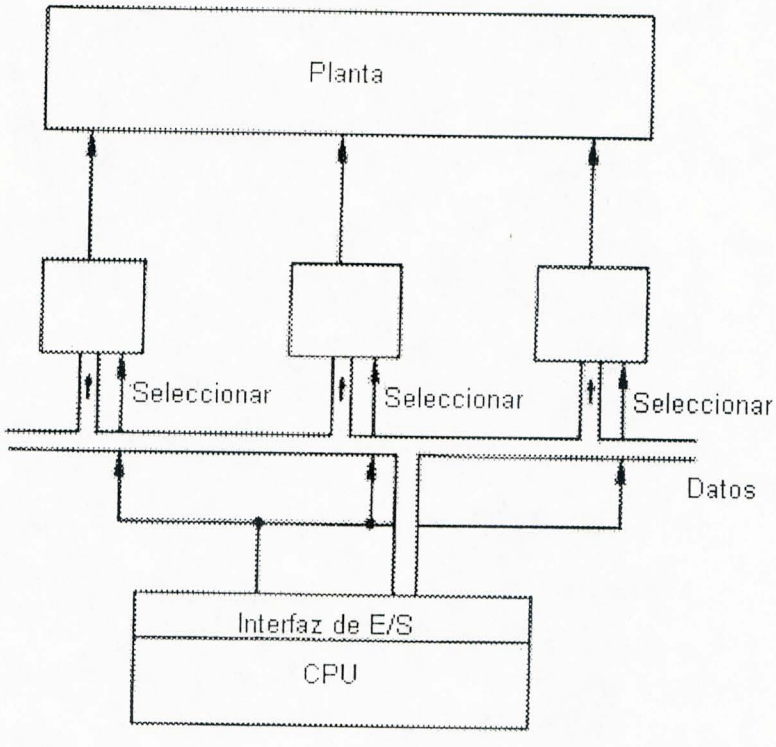


Figura 5
Conversor Digital/Analógico

1.2.3 Control Secuencial

Las técnicas de control secuencial son tan importantes como las de Control Retroalimentado. A pesar de que hay acumulada mucha experiencia práctica, su sistematización y su base teórica están muy poco desarrolladas comparadas con las de control retroalimentado.

El control puede ser definido como la aplicación de una acción pre-planeada, para que aquello que se considera como objeto satisfaga ciertos objetivos [Miy93]. En el caso del control secuencial, corresponde a la ejecución de operaciones paso a paso, según una secuencia preestablecida o una lógica fija que establece la secuencia. En este tipo de control no existe el concepto de valor de referencia, éste ha sido sustituido por el de comando de control. Estos comandos son en general señales digitales, que regulan la evolución de los pasos de la secuencia. La evolución de estos pasos dependen de que cada estado cumpla todas condiciones de las reglas preestablecidas para pasar al paso siguiente. Las condiciones que regulan esta evolución son de dos tipos: las que dependen del tiempo (time driven), donde las condiciones para la evolución pueden representarse a través de funciones de tiempo y las que dependen de eventos externos (external event driven), donde las condiciones para la evolución pueden ser representadas a través de señales de entrada externas.

El control secuencial se usa generalmente para controlar sistemas de procesamiento en lote como es el caso del procesamiento de alimentos y de las industrias químicas. A modo de ejemplo analizaremos la secuencia de control de un tanque de medición: un sistema para medir un cierto volumen fijo de líquido a través del control del nivel del líquido. Cuando el nivel del líquido está en el mínimo, las válvulas que permiten la entrada y salida del material permanecen en modo cerrado y se enciende la lámpara que indica el fin del proceso. Esta lámpara también indica que el sistema está listo para realizar la próxima medición. Presionando el botón de accionamiento en estas condiciones, la válvula de acceso del material se abre, permitiendo el paso del líquido al interior del tanque, y se inicia el proceso de medición. La llegada del líquido al nivel máximo es detectada por un sensor. Con esto la válvula de entrada se cierra y se abre la válvula de salida, permitiendo que salga el líquido. Al llegar al nivel inferior, se activa el segundo sensor (que indica el líquido llegó al nivel mínimo), cerrando la válvula de salida.

En este caso la secuencia da pasos es la siguiente:

1. Estado inicial. Es el estado de reposo del sistema, es decir no hay flujo de material.
2. Este estado indica el inicio de la medición. O sea el nivel del líquido es menor que el mínimo y por lo tanto existe flujo de entrada del material y no existe flujo de salida.
3. Este estado indica que el nivel del material es intermedio. Sólo existe flujo de entrada.
4. Este estado indica que el material alcanzó el nivel máximo. No existe flujo de entrada ni de salida del material.
5. Estado que indica el inicio de la salida del material. Sólo existe flujo de salida
6. Estado que indica que el proceso de salida del material está ocurriendo, o sea que el nivel del líquido en el tanque es intermedio. Sólo existe flujo de salida

1.3 Supervisión de Procesos

Un supervisor es un conjunto de programas encargados de coordinar, monitorear y prestar servicio a los diversos componentes del sistema. El supervisor deberá controlar la entrada y salida de datos. También será el encargado de planificar las tareas que el sistema debe llevar a cabo, establecer prioridades entre los programas de aplicación, procesar errores y condiciones de alarmas.

Una aplicación que procesa datos en forma convencional, usualmente lo hace siguiendo un ciclo repetitivo de eventos, los cuales fueron planeados en detalle por un programador. En un sistema en tiempo real esto no es posible. Los mensajes pueden llegar en cualquier momento y probablemente ser de distinto tipo y longitud, y la secuencia de operaciones no se puede predecir. A pesar de esto, los datos deberán ser procesados a tiempo y haciendo uso eficiente de los recursos.

Los sistemas en tiempo real, no pueden construirse con un patrón predeterminado de eventos. Una falla en alguno de los componentes de hardware del sistema, necesitará procedimientos de emergencia, y éstos deben ser automáticos. A su vez deben existir funciones que les permitan detectar comportamientos anómalos, deben poder comunicar estos estados de emergencia al operador (a través de alarmas o mensajes), y permitir intervenir al operador para solucionar el problema. En algunos casos deben proveer soluciones automáticas.

Por todo esto se necesitan sistemas de Supervisión de Procesos que permitan planificar el trabajo, asignar espacio de almacenamiento, determinar prioridades y procesar estados de emergencia, si es que esto no puede ser predeterminado por el programador. Además deben proveer un conjunto de herramientas poderosas para que el operador puede mejorar su trabajo en la planta.

Por otra parte, el uso de computadoras incrementó el rango de actividades que pueden realizarse, de manera tal que no solo son usadas para realizar el control de la planta, sino también proveen a los

ingenieros y gerentes de un informe completo de la operación de la planta, a través de informes y gráficos de distintos niveles.

En general los Supervisores tienen funciones comunes, independientemente de la aplicación que están supervisando. Es por este motivo, es en ese campo donde se están introduciendo la mayoría de los estándares para sistemas en tiempo real y en donde se realizaron la mayor cantidad de cambios y avances. En la siguiente sección profundizaremos cuales son las principales funciones que deben ser provistas por los supervisores en general y a continuación ejemplificaremos con algunas aplicaciones existentes.

1.3.1 Funciones de Supervisión

Tradicionalmente se utilizaron PLCs (programmable logic controllers) para el control en tiempo real y la PC para las funciones de supervisión y adquisición de datos. Esto se debió a que en un comienzo los PLCs eran más económicos que las mini computadoras, más confiables y más fáciles de usar. Las razones del éxito de los PLCs se debió básicamente a que la construcción de los mismos se basó en componentes altamente confiables de estado sólido, y a que los circuitos electrónicos fueron diseñados especialmente para entornos industriales.

Otra razón importante es que en un principio el lenguaje de programación elegido para los PLCs fue RLL (Relay Ladder Logic), un lenguaje basado en la lógica de circuitos relé, que resultó fácil de aprender a los ingenieros y técnicos de la planta, quienes por el contrario, no tenían ningún conocimiento de programación de computadoras. Por otro lado el equipamiento era caro y no se justificaba una inversión de este tipo para ser usada en lugar del equipamiento analógico que ya se poseía.

Por lo tanto las computadoras quedaron relegadas a un rol de supervisión, o sea los PLCs realizaban el control de la planta (usualmente son varios PLCs involucrados en el control de la planta), y la computadora se programaba para supervisar las operaciones y enviar comandos a los PLCs para tomar alguna acción correctiva sobre el medio. Podemos ver un ejemplo de supervisión en la Figura 6.

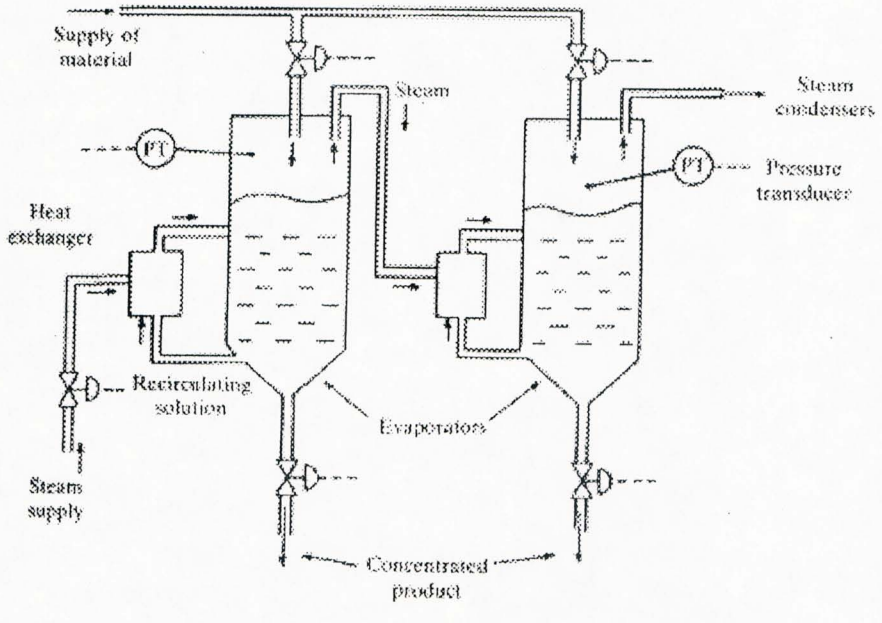


Figura 6
Planta de evaporación

En este diagrama vemos dos unidades de evaporación conectadas en paralelo, cada unidad es alimentada con una solución de materiales. El propósito de la planta es evaporar la mayor cantidad de agua posible de la solución. Para esto la primera unidad tiene conectado un dispositivo para intercambiar calor, al que se le suministra vapor. El vapor para la segunda unidad es suministrado a partir del vapor producido por ebullición en la primera unidad. Para alcanzar la máxima evaporación, la presión en las cámaras deberá ser lo mas alta que la seguridad permita. Sin embargo es necesario alcanzar un balance entre las dos unidades de evaporación: si el primero es llevado al máximo de su capacidad, podría generar tanto vapor que esto haga que el límite de seguridad de la segunda unidad sea excedido.
En este caso, podría adoptarse un esquema de supervisión para balancear la operación de las dos unidades de evaporación, y así la mejor tasa de evaporación. Este supervisor deberá monitorear las dos unidades en tiempo real. Es decir, deberá ser capaz de obtener y procesar información del ambiente, detectar cualquier anomalía y tomar alguna acción en forma inmediata.

Para simplificar la organización de estas tareas, todos los supervisores suelen manejar un conjunto mínimo de información, que está definido por una transacción de cambio del estado de las variables del ambiente. Esta mínima unidad de información suele conocerse como *lazo*, y está representado por un conjunto de variables y acciones que describen los atributos del sistema que se quiere monitorear.

El concepto mas importante de un lazo es que puede ser usado para estructurar el proceso de desarrollo del supervisor. Podemos imaginar a los distintos lazos como funciones independientes del sistema que deben ser integradas. Por lo tanto un Supervisor, no es otra cosa un conjunto de tareas que permite integrar el análisis de varios lazos y que permite obtener un diagnóstico de la evolución de los estados del sistema para estos lazos.

Consideraremos como ejemplo un sistema de control y supervisión de un edificio. Podemos encontrar los siguientes atributos: alarmas contra invasores, alarmas contra incendios, control ambiental, control de acceso a determinados lugares del edificio. Para cada uno de estos atributos existen sensores que leen el estado de las variables en cuestión y se comunican con alguna estación de supervisión. Por ejemplo en el caso de un lazo de incendio, tendremos un conjunto de detectores de humo y de temperatura que alimentan a un conjunto de variables. Tales valores deben ser comunicados a los puestos de monitoreo a través de drivers especiales. Allí son analizados para verificar si realmente se está produciendo un incendio y evitar falsas alarmas debido a focos insignificantes. En caso de confirmarse el incendio, el paso siguiente sería el accionamiento de alarmas y la activación de procedimientos especiales, tales como lanzamiento de agua y espuma, poner fuera de funcionamiento los ascensores, emisión de instrucciones verbales previamente grabadas para evacuar el predio en forma ordenada, etc., dependiendo del grado de complejidad de la automatización del edificio. Similarmente pueden definirse lazos de seguridad, ambientación, etc. Podemos imaginar los lazos como funciones independientes de un sistema, que deben ser integradas.

Si tomamos a un lazo como a un conjunto de variables, y no tomamos en cuenta los equipos que son utilizados para obtener sus valores directamente de la planta, entonces un sistema supervisor es un ambiente computacional que integra modelos de varios lazos. Un estado del sistema es un conjunto de n-uplas, conteniendo los valores para cada lazo.

Por otro lado, podemos identificar un conjunto de variables que integran cada lazo. Las variables envueltas en los lazos pueden ser de varios tipos, pero en general, se aceptan, como mínimo, variables analógicas y variables digitales. El conjunto de todas las variables del sistema supervisor, conforman una base de datos, y cada componente de la misma se conoce con el nombre de *punto de supervisión*. La función primordial de estos puntos es el monitoreo de variables del proceso (temperatura, presión, flujo, nivel), y por lo tanto se deben realizar muestreos periódicamente. Los puntos además pueden usarse para control, despliegue de datos, ejecución de procesos asociados y cualquier otra función de supervisión.

La utilización de una base de datos central es primordial en todo sistema supervisor, ya que permite almacenar todas las variables de los lazos. Esta base de datos puede ser usada por las tareas que manejan el equipamiento de la planta, por las rutinas de conexión del supervisor con otros sistemas, y por el propio sistema supervisor, ya que todas las variables de los lazos están almacenadas en ella. Muchas de las aplicaciones de supervisión son muy sencillas y se basan en el diagrama de estado de la planta. Las aplicaciones mas complejas hacen uso de técnicas mas complicadas como programación lineal y simulación, que incluye modelos no lineales complejos de la dinámica de la planta. En estas aplicaciones, los algoritmos complejos, deben ser procesados en tiempo real en paralelo con la operación de la planta.

Analizando con cuidado las funciones que deben ser provistas por un sistema de supervisión de procesos, debemos considerar las siguientes :

- *Funciones de Supervisión del Sistema.* Son funciones que permiten al operador monitorear el sistema. Por ejemplo el sistema despliega en pantalla los valores de los distintos lazos, esto le permite al operador de la planta tener información directa y concisa del estado de la planta. Entre estas funciones también se incluyen aquellas que permitan al operador cambiar el estado de la planta a través de comandos ingresados en la computadora.

- *Funciones de Validación.* Implica la validación de datos de control chequeo del correcto funcionamiento de los equipos y la administración de estos datos, funciones todas que tratan de mantener y asegurar la consistencia y correctitud de los datos que se transmiten desde la planta a la computadora.

- *Funciones de registro de operaciones del sistema y funciones de análisis.* Las funciones de registro de datos permiten mantener la historia del proceso, lo que ayuda a detectar la causa de un error en el sistema o en los equipamientos. Esto se logra registrando la operación del sistema en todo momento. Los datos que mas comúnmente se guardan son :

  1. Cambios en el estado del equipamiento
  2. Registros de errores, para reconocer los estados cuando el equipo se encuentra defectuoso y así evitar fallas posteriores
  3. Registros de operaciones, es decir guardar las operaciones realizadas por el operador o la computadora para controlar el sistema en un determinado momento.
  4. Registros de estados que indiquen el estado del sistema

  Adicionalmente al registro de los datos, deberá existir alguna forma de recuperar estos datos y crear informes, de acuerdo a los propósitos del usuario, brindándole funciones como resumen de datos, cálculo de valores promedio, varianzas, y desvíos estándar y ordenamiento.

- *Funciones de Administración general del sistema.* Incluyen, atención de interrupciones, manejo de excepciones en el sistema, administración de colas y mensajes, así como también planificar y dar prioridades a las distintas tareas que se deben ejecutar para el buen funcionamiento de la planta.

Con respecto al análisis de los datos y tratamiento de casos de excepción, podemos ver que, los sistemas mas simples dejan toda la tarea de análisis para el operador y los mas complejos poseen mecanismos de inferencia que brindan apoyo para la toma de decisión.

Los sistemas de supervisión también se conocen como sistemas SCADA (Supervisory Control and Data Adquisition). Un sistema SCADA es un ambiente computacional, capaz de integrar y analizar diversos lazos, de modo de obtener un diagnóstico de la evolución de los estados del sistema, y también actuar sobre esos estados en caso de ser necesario.

Un punto clave en la adopción de un sistema SCADA es el nivel de facilidades que el sistema provee al operador de la planta. Es importante que este cuente con una interfaz simple y clara para la operación diaria de la planta. El sistema debe ofrecer facilidades para cambiar set points, ajustar variables, controlar condiciones de alarma, etc. Muchos productos ofrecen la posibilidad de presentar información en planillas de cálculo, que pueden editarse por celdas, que facilita la visualización de la información.

Los SCADA, en general, proveen distintos niveles de información en su interfaz hombre/máquina:

- para el operador de la planta, displays gráficos mostrando estados de alarmas, displays presentando información de distintas áreas de la planta y facilidades para interactuar con la planta.
- para el ingeniero de la planta, incluyendo gráficos de tendencias y resúmenes de operaciones pasadas, así como también información para la toma de decisiones, asociadas con el mantenimiento de la planta y reemplazo de componentes.
- para el gerente de la planta, acceso a cierto tipo de información, en la forma de informes (incluyendo gráficos), que resuman la operatoria diaria de la planta, informes que presenten datos históricos e informes estadísticos, que facilitan el seguimiento de la operatoria y permiten determinar mejoras.

Cada usuario del sistema tendrá definido su perfil, es decir, qué operaciones les están permitidas realizar sobre la planta, y por lo tanto que información deberá brindarle el sistema para cumplir con sus tareas.

1.3.2 Arquitectura de un sistema SCADA

Como vimos en los párrafos anteriores un sistema SCADA es un sistema que permite monitorear y controlar procesos. En particular nosotros estamos interesados en sistemas donde el tiempo necesario para la actualización de un conjunto de variables, a partir de valores extraídos del mundo real, tenga un máximo determinado.

Por lo tanto los lazos deben tener un tiempo característico determinado para el proceso en sí, y para el análisis y monitoreo de los resultados. Otro factor importante es la existencia de una Base de Datos (centralizada o distribuida) que contenga todas las variables de los lazos para que puedan ser accedidas por los procesos que conectan el sistema con la planta (drivers de entrada/salida), por las rutinas de interfaz que conectan al ambiente con otros sistemas, y por otros módulos como planillas de cálculo o sistemas inteligentes para el análisis del proceso.

La Figura 7 muestra un esquema general de un sistema SCADA.

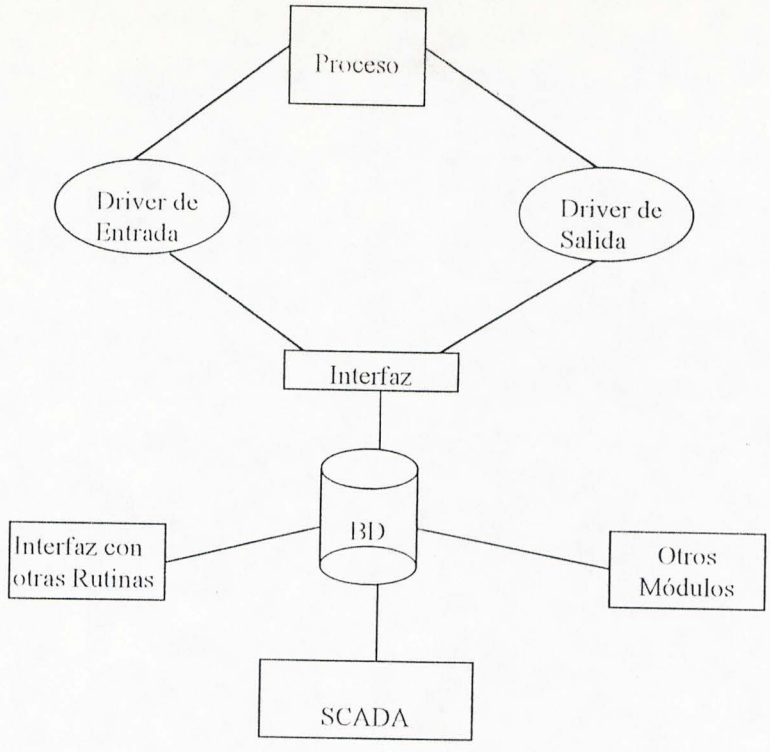


Figura 7
Arquitectura de un Ambiente de Supervisión

Allí vemos una base de datos centralizada que es accedida por los diferentes procesos. De esta manera se evita que se produzcan redundancias en la base de datos, como por ejemplo que una misma variable tenga valores diferentes en un mismo instante (uno proveniente de la planta y otro ingresado por el operador de la misma).

Algunos SCADA son bastantes flexibles como para permitir que, a través de interfaces apropiadas, puedan utilizar bases de datos comerciales (generalmente relacionales). Esto permite utilizar los mecanismos de consistencia e integridad de datos de estas bases de datos.

Otro aspecto importante es realizar un buen modelado del sistema. Esto permitirá detectar cuales son los datos relevantes para monitoreo y supervisión, ya que el monitoreo de variables innecesarias comprometería eficiencia y tiempo de respuesta. La automatización pretende justamente eliminar redundancias de este tipo, ya sea construyendo sistemas que sustituyan el elemento humano en estas tareas, estructurando el proceso de interacción entre sistemas y operadores.

Una ventaja de tener un modelado preciso del sistema es que permite la verificación del mismo a través de simulaciones. A pesar de que esto no es el objetivo primordial de un ambiente de monitoreo y supervisión, algunos sistemas conocidos disponen de herramientas de simulación que pueden ser útiles en la predicción de fallas y en la verificación del funcionamiento de alarmas y de rutinas de emergencia. Otra ventaja adicional es que la presencia de un módulo de simulación facilita el entrenamiento del personal para la utilización del sistema.

El análisis y tratamiento de casos de excepción en el funcionamiento de la planta es un objetivo principal en ambientes de supervisión. Los sistemas SCADA pueden ser clasificados de acuerdo con los recursos de que disponen para el análisis de estados críticos: los sistemas simples dejan toda la tarea de análisis para el operador; los sistemas inteligentes poseen mecanismos de inferencia para ayudar en el proceso de análisis, y sistemas de apoyo en la toma decisiones, para auxiliar al operador a seleccionar las rutinas de excepción.

La mayor parte de los sistemas SCADA disponibles en el mercado, dejan el análisis a cargo del operador. Un pequeño número utiliza el modelado formal para guiar al operador acerca de qué puede esperar del comportamiento del proceso, sin comprometer el tiempo de respuesta con mecanismos de inferencia basados en la lógica. Algunos sistemas utilizan modelado a través de objetos, con sistemas de apoyo en la toma de decisión basados en reglas (sistemas de producción).

Aún es prematuro hablar de metodologías de desarrollo de aplicaciones de sistemas SCADA, pero es posible observar algunas pautas, corroboradas por la práctica de estos sistemas, principalmente en control de procesos de plantas químicas. Generalizaremos estas pautas para cualquier supervisor.

En primer lugar, es importante resaltar que estamos hablando de sistemas donde el papel del sistema SCADA no es auxiliar, ya que existen controladores que le permiten comunicarse con la planta de forma independiente. De esta manera el papel del supervisor es verificar el comportamiento de la planta y de mantener este comportamiento dentro de ciertos parámetros predefinidos a través de la comunicación con los controladores. Por lo tanto los sistemas SCADA pueden ser sustituidos sin que esto signifique un cambio drástico a la aplicación.

Otra característica importante, es que podemos tratar a los supervisores como un sistema de software dejando para mas adelante los problemas de implantación. Es decir podemos ver a un SCADA como un sistema integrador de los controladores asociados a la planta.

Con este enfoque, el desarrollo de aplicaciones SCADA es análogo al desarrollo de cualquier otro sistema. Por lo tanto en una primera fase debemos modelar el proceso, identificando las variables y propiedades relevantes que representen el funcionamiento de acuerdo a las características generales de la aplicación.

Es importante también poder discernir si una variable introduce información sobre la planta (variable de entrada) o si al contrario, es usada para interactuar con el proceso (variable de salida). Repitiéndose el proceso de definición de variables para cada uno de los lazos, podremos tener una idea precisa de como se comparten las variables, esto es si un mismo dato se comparte en lazos diferentes. Las redundancias deben ser eliminadas cada vez que sea posible, para evitar duplicación de drivers y/o transductores.

El siguiente paso en el desarrollo de sistemas SCADA, será distinguir las precondiciones para el funcionamiento del sistema. En caso de que alguna de las precondiciones no sea satisfecha, se deberán accionar alarmas, mandar mensajes a los operadores, etc. Una vez definidas las precondiciones, se deberán definir las interfaces apropiadas para cada operador, presentándole a cada uno la información que necesite para poder operar la planta. Deberán definirse distintos perfiles, con distintos niveles de autorizaciones para operar sobre la planta. Con todo esto definido y programado podemos simular el funcionamiento del sistema con entradas programadas o aleatorias. Análogamente al caso de la construcción de software, es importante la programación de todos los casos que van a requerir el accionamiento de alguna alarma.

Si la base del sistema SCADA funciona en forma aceptable, entonces, podemos iniciar la programación de las partes mas complejas, como por ejemplo la elaboración de programas basados en técnicas de inteligencia artificial (si el sistema lo permite y si la aplicación justifica este esfuerzo). En sistemas orientados a objetos, las variables son definidas como atributos de una clase. Dicha clase representa a cada lazo. En este caso podríamos dejar para el final la definición de los métodos y rutinas de emergencia. Todo el conocimiento asociado a la aplicación puede ser guardado en una base de conocimiento denominada dominio de la aplicación, para ser utilizada por el sistema, usando técnicas de inteligencia artificial (reglas de inferencia).

Tomemos como ejemplo un sistema de supervisión y monitoreo de un edificio. En estos sistemas domóticos, el análisis y diagnóstico en general son simples y son dejados el operador. La función

principal del sistema supervisor es facilitar y agilizar el acceso a variables de análisis, de modo que el tiempo de respuesta al operador sea óptimo.

Sin embargo, a partir de ese conjunto básico de lazos, no es posible hacer un análisis mas elaborado, evitando redundancias y aumentando la integración de las distintas partes del sistema. Por ejemplo la ambientación podría contar con un funcionamiento cooperativo en una torre central, donde se recibe la información de las distintas áreas del edificio. Este funcionamiento cooperativo podría tener como objetivo optimizar el consumo de energía, y también eliminar las variaciones bruscas en la temperatura de distintos ambientes. Esto no pude ocurrir si los sistemas de aire acondicionado y circulación trabajan en forma localizada, sin tener en cuenta el efecto combinado.

En casos como este, pueden ser útiles sistemas basados en conocimiento y/o orientados a objetos, para proveer un proceso de análisis y diagnóstico mas elaborado del proceso. También pueden ayudar a hacer aún mas eficiente la acción del operador, proveyendo un diagnóstico de fallas, apoyo de decisiones, etc.

En general las herramientas basadas en conocimiento que se encuentran en sistemas supervisores, se resumen en sistemas expertos, cuya arquitectura simplificada podemos ver en la Figura 8.

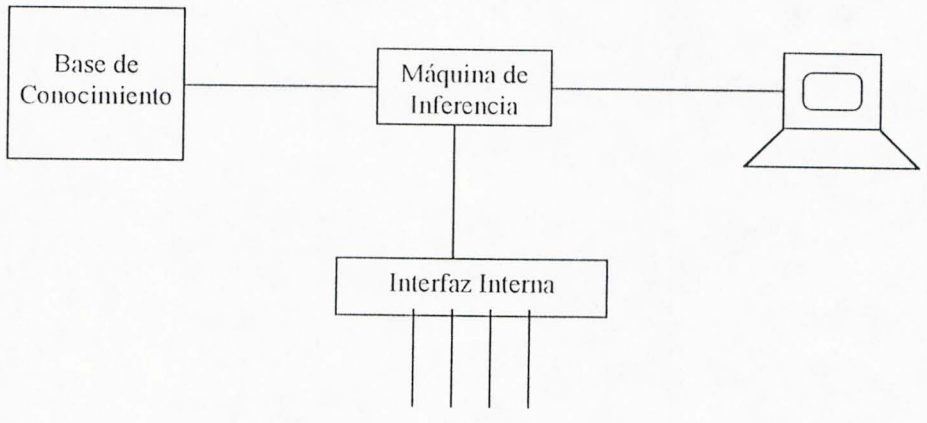


Figura 8
Modelo de un Sistema Experto

La parte mas importante de un sistema experto es la base de conocimiento. Esta base debe contener toda la información sobre el sistema. En el caso de un sistema supervisor esta base de conocimiento contendrá toda las reglas que describen los estados del sistema. El motor de inferencia esta compuesto por las reglas de deducción, esto es una secuencia de reglas capaces de inferir cuales son las transformaciones de estado admisibles. En el caso de los supervisores esta herramienta es utilizada para analizar el proceso.

La interfaz externa maneja la interacción del sistema con la estación de trabajo, donde se puede visualizar el estado del proceso. La función de esta herramienta es transformar en mensajes, alarmas, etc., el procesamiento simbólico.

La interfaz interna transforma las variables simbólicas, reglas, hechos etc., en lecturas de las variables del proceso. En esta interfaz se encuentra una parte considerable de la interpretación del modelo del proceso. De esta forma se puede trabajar con evaluaciones cualitativas del proceso, y relacionar varios lazos diferentes, con el fin de realizar algún análisis.

Algunos sistemas SCADA están basados en tecnología orientada a objetos. Esto significa que si el sistema necesita un dato, puede requerirlo a través de la llamada a una función, y este dato será traído de cualquier lugar donde se encuentre. La aplicación no necesita saber donde o como está almacenado el dato, el objeto es quien se encarga de esto y devuelve el dato de manera que pueda ser usado por quien lo requirió.

Cada objeto es un conjunto de funciones, denominadas métodos y de atributos. Los métodos pueden ser públicos o privados. Los privados son usados solo por el objeto para realizar su trabajo. Esto incluye la representación interna y externa de los datos. Los métodos que son públicos pueden ser usados por otros objetos o funciones en la aplicación para acceder la información que el objeto posee. El hecho de tener áreas públicas o privadas, permite al objeto cambiar su representación interna de los datos, y los algoritmos, para mejorar los tiempos de procesamiento, sin por ello afectar a los métodos que son públicos. Esto significa que los cambios a un objeto pueden hacerse internamente sin afectar a la aplicación misma.

En algunos sistemas la base de conocimiento del supervisor es modelada por instancias de clases, que a su vez son representaciones asociadas al dominio de conocimiento de la aplicación. Así por ejemplo para aplicaciones en el área química, podría definirse una *Clase_Tanque:*

Clase_Tanque ::= { Nombre = string(20);
Función = (mezcla, almacenamiento, etc.);
Area_de_la_base = real;
Altura = real;
Nivel_máximo = real;
Nivel_mínimo = real; }

Esta clase definiría a todos los posibles tanques relacionados con la aplicación. Luego, por ejemplo un tanque que se use para mezclar elementos, sería una clase derivada de la clase tanque :

Clase_Tanque_para_ mezclar ::= { Nombre = string(20);
Función = (mezcla, almacenamiento, etc.);
Area_de_la_base = real;
Altura = real;
Nivel_máximo = real;
Nivel_mínimo = real;
Nivel_de_operación = real; }

La clase "tanque para mezclar", hereda todas las propiedades de la clase tanque y tiene un atributo mas, el nivel de operación del tanque, que es el nivel mínimo para poner en funcionamiento la mezcladora.

Una instancia T1 de esta clase se definiría por el siguiente objeto :

T1 = { Nombre = T1;
Función = mezcla;
Area_de_la_base = 1.002 m2;
Altura = 1.0 m;
Nivel_máximo = 0.95 m;
Nivel_mínimo = 0.05 m;
Nivel_de_Operación = 0.22 m }

La introducción de objetos es importante porque agrega el concepto de reutilización de modelos y permite asociar a cada implantación en particular clases distintas propias del proceso que se desea modelar. Es también posible asociar a cada clase una representación gráfica, en este caso todos las instancias de esta clase tendrán la misma representación gráfica.

En la mayoría de las aplicaciones la herencia simple, (una clase hereda los atributos de otra clase madre, y acrecienta los atributos con atributos específicos), es suficiente para representar los procesos que se desean monitorear. Pero es importante notar que la programación orientada a objetos también posee mecanismos para relacionar lazos, ya que los objetos pueden heredar atributos de distintas clases (herencia múltiple). Esta propiedad es muy importante para evitar redundancias y definiciones repetidas.

En el desarrollo orientado a objetos se comienza por la identificación de las clases de objetos básicas y de sus propiedades específicas. En caso de un desarrollo orientado a objetos la metodología utilizada es bottom-up, partiendo de la definición de objetos y componiéndolos para obtener el modelo del proceso deseado.

1.3.3 Categorías de SCADAs

Según un análisis realizado en [Kap95], pueden definirse 5 categorías de SCADAs teniendo en cuenta facilidad de uso y flexibilidad del software: Software específico, Interfaz con un lenguaje, Facilidades agregadas, Código Fuente e Instrumentación Virtual. Para ello se consideraron todos los tipos de sistemas que tengan interfaz con hardware para adquisición de datos. Se asume como punto de partida que este hardware debe tener embebido el código necesario para configurar los registros para realizar la entrada/salida analógica y digital y la sincronización. También se asume que el software debería controlar la transferencia de datos desde y hacia el hardware especializado, ya sea usando polling, interrupciones o acceso directo a memoria.

Software Específico. Es el mas fácil de usar y necesita muy poca preparación seteo inicial. Este tipo de software está diseñado especialmente para controlar una aplicación en particular. Si el usuario necesita funcionalidad adicional a la incluida, deberá convencer al proveedor del software de que dicha funcionalidad será agregada en el paquete original.

Interfaz con un lenguaje. Es una colección de subrutinas o llamadas a funciones desde lenguajes de programación convencionales como por ejemplo Pascal, Basic y C. Mientras que el Software específico, es orientado a usuarios finales y no a programadores, el software de interfaz con un lenguaje se orienta a personal de desarrollo que puede programar en algún lenguaje convencional. Los usuarios de este tipo de software deberán ser capaces de escribir, compilar código y linkeditarlo con la interfaz provista para poder realizar las tareas de adquisición de datos. Con la ayuda de esta interfaz, los programadores pueden acceder al hardware de adquisición de datos a través de simples llamadas a funciones. Una vez que los datos fueron recolectados y almacenados en memoria, los programadores deberán realizar el resto de la programación para el manejo y presentación de esos datos, o utilizar alguna herramienta de análisis y visualización.

Facilidades agregadas. Son funciones incluidas en algún entorno de desarrollo conocido, como por ejemplo Lotus 1-2-3 o Excel. Estas herramientas permiten direccionar los datos obtenidos a través de los ports a estas planillas de cálculo (u otra aplicación), y visualizarlas en forma de gráficos etc.

Código fuente. Son archivos, usualmente escritos en C, que los programadores pueden compilar con la aplicación para adquirir y controlar datos. Este código fuente es usualmente utilizado como un ejemplo de cómo debe realizarse la programación de las plaquetas a nivel de registro. Esta es la forma más compleja de programar un SCADA, y la mas lenta para obtener un sistema funcionando. A pesar de esto, ayuda a reducir el tamaño del software final. Sin embargo en la actualidad, este tipo de software está perdiendo terreno, debido a que el hardware para adquisición de datos es mas poderoso y mas difícil de programar. Para hacer esto último el programador no solo debe estar familiarizado con el lenguaje de programación, sino también con la operación de interrupciones, acceso directo a memoria y ports de E/S de la PC.

Instrumentación Virtual. Permite a los usuarios diseñar los instrumentos mas apropiados para su aplicación. La Instrumentación Virtual usa el hardware de adquisición de datos disponible y de la PC. Realiza las interfaces entre ambos utilizando los estándares de arquitecturas abiertas para procesamiento, almacenamiento en memoria y display de la información. El usuario define la funcionalidad de los instrumentos virtuales, permitiéndosele agregarle formas elaboradas de control.

1.3.4 Algunos SCADA de ejemplo

Quince años atrás el uso de sistemas SCADA (Supervisoy Control and Data Adquisition) era exclusivo de los científicos que trabajaban con lenguajes tales como C y Fortran o lenguajes de aún mas bajo nivel, como por ejemplo assembler, escribiendo programas para aplicaciones específicas, cuya plataforma eran las mini computadoras. Hoy en día el software para SCADA ha evolucionado notablemente, siendo no solo mas fácil de usar, sino también mas poderoso. Estos adelantos permiten que su uso escape al ámbito científico y pueda ser utilizado por personas no expertas para automatizar las funciones de supervisión y adquisición de datos.

Hoy pueden encontrarse en el mercado una amplia gama de software SCADA que varían desde paquetes sumamente flexibles y adaptables como código fuente especial para adquisición de datos (que deberá ser compilado dentro de la aplicación), hasta paquetes poco flexibles pero fáciles de usar como Instrumentación Virtual. Analizaremos a continuación algunos de ellos.

1.3.4.1 TAURUS

Es un paquete de programas integrados, totalmente gráfico e interactivo, operable a través de menúes configurables por el usuario. Este SCADA es una facilidad agregada a otras herramientas como Excel 5.0 [Kap95]. Dentro del paquete se incluyen herramientas para la generación y manipulación de gráficos y mímicos. El sistema TAURUS se encuentra estructurado como se muestra en la Figura 9. Está compuesto por módulos o ambientes de trabajo, algunos de los cuales pueden ejecutarse fuera de línea (off-line), independientemente de si el sistema está ejecutándose en tiempo real o no.

TAURUS posee una interfaz gráfica configurable por el usuario, orientada a facilitar la tarea del operador. Dentro del paquete básico se incluyen herramientas para la elaboración de pantallas y de mímicos que permiten representar la dinámica de la planta. El sistema puede conectarse con cualquier instrumento o programa existente para entrada y salida de datos, ya que viene provisto de una variedad de drivers de comunicación, ya sea para equipos de planta (PLC, micro controladores, placas de adquisición de datos, etc.) o para conexión en red de dos o mas TAURUS. Los datos obtenidos pueden presentarse en pantalla para su visualización u operación.

Otra característica importante es que provee un completo panel de alarmas, que informa, a través de un código de colores los estados de las mismas, indicando además el tipo de alarma producida. Todos los cambios producidos en la planta son guardados en archivos históricos, lo que permite al usuario disponer de esta información para su uso directo o para procesamiento posterior (como por ejemplo reportes de tendencias).

TAURUS acepta la definición de variables en una planilla, conteniendo varios campos predefinidos que corresponden a los campos de los registros en la base de datos central. Permite el manejo de variables digitales ó analógicas y además variables de tipo caracter.

Como vemos en la Figura 9, TAURUS está compuesto por módulos independientes. Podemos agrupar estos módulos en: módulos orientados a la *operación de la planta y análisis de información* y módulos orientados a la *configuración y diseño de aplicaciones*. Los módulos orientados a la *operación de la planta y análisis de información son:*

- *Taumotor*. Es un programa residente cuya función es la adquisición datos y el control. Se ejecuta concurrentemente con los demás procesos
- *Taugint* que es un programa gráfico interactivo, para la operación de la planta. Una vez configurada una aplicación en particular, este será el ambiente en donde trabajará el operador de la planta. La información con la que opera Taugint (analógica y digital), es suministrada por el Taumotor. A través de esta interfaz el operador podrá manipular los instrumentos, indicadores y controladores que hacen al funcionamiento de la planta.
- *Tauhist*. Este módulo realiza el análisis de los datos históricos, generados por Taumotor. Muestra estos datos en forma gráfica y numérica con posibilidad de salida impresa, cálculos estadísticos e integrales, exportación de archivos a otros programas.

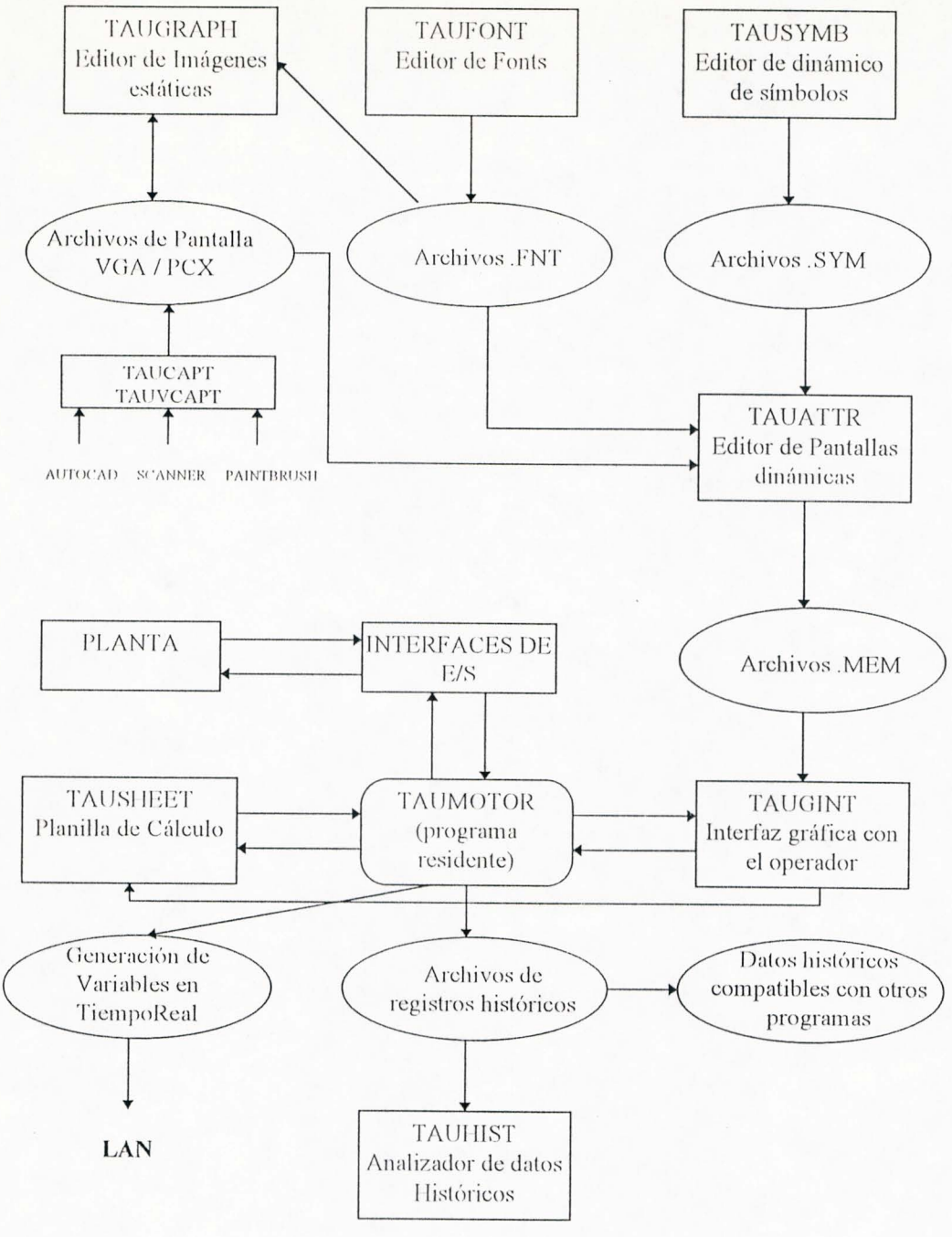


Figura 9
Diagrama de Estructura de TAURUS

Los módulos orientados a la *Configuración y diseño de aplicaciones son:*

- *Tausheet.* Planilla de cálculo editable por celdas. Permite la visualización de los datos históricos y los datos de tiempo real.
- *Taugraph.* Programa para la generación de gráficos de gran resolución. También permite la captura y modificación de gráficos generados por otros programas (por ejemplo CAD), imágenes ingresadas a través de scanners, imágenes digitales capturadas mediante vídeo cámaras. Esta captura, se realiza a través del programa auxiliar Taupcapt.

- *Tauattr.* Este módulo permite diseñar atributos dinámicos de la pantalla. Sobre un fondo estático diseñado mediante Taugraph, pueden definirse iconos para ejecutar series de comandos, cambiar a otras pantallas, etc.
- *Tausymb.* Se utiliza para la creación de iconos, que luego pueden ser incorporados a las pantallas a través de Tauattr, y operar los mismos en el ambiente Taugint. Estos iconos pueden variar de tamaño, color o posición de acuerdo a una variable del mundo real a la que están relacionados, lo que permite la creación de mímicos con movimiento que simulan con realismo la dinámica de la planta.
- *Taufont.* Programa para la creación de tipos de letra para textos, con tamaños que pueden ser seleccionados de hasta 30x32 pixels. También pueden capturarse tipos de letra de alguna biblioteca de tipos de letras, y luego ser procesados.

En cuanto al hardware requerido para instalar TAURUS es necesario contar con una PS o PC compatible, con microprocesador 286 (o superiores), con un mínimo de 640 Kbytes de memoria RAM y un mínimo 40 Mbytes de disco rígido. No es necesario pero si aconsejable que cuente con un coprocesador matemático. Deberá tener un monitor y placa gráfica VGA, así como también una placa RS-232 (mínimo) y una placa paralelo (mínimo).

1.3.4.2 ONSPEC

Es un sistema SCADA de avanzada, diseñado para trabajar con dispositivos para adquisición de datos y para proveer monitoreo y control en tiempo real, en cualquier entorno de proceso. Es un SCADA dentro de la categoría de Código Fuente e Instrumentación Virtual según [Kap95]

El propósito de ONSPEC es ayudar al ingeniero a comprender el proceso en el que trabaja. Para ello la Base de Datos central toma datos del mundo real, los almacena en tablas internas, para luego mostrarlos gráficamente, analizarlos y controlarlos. La interfaz de entrada/salida actualiza la tabla con las lecturas del mundo real. Se compone de 3 elementos principales :

1. El programa ONSPEC es el corazón del sistema. Este programa manipula los datos que representan el estado de la planta. Estos datos son presentados en forma gráfica, monitoreados para reconocer condiciones de alarma, y controlados por ONSPEC.
2. La interfaz de Entrada/Salida, es un conjunto de programas (drivers) separados de ONSPEC, y es la responsable de la comunicación entre ONSPEC y el mundo real. La interfaz de Entrada/Salida, recolecta datos del exterior, a través de un port serial de comunicación, y pasa los datos a ONSPEC. Esta interfaz, también permite comunicar al mundo real, los ajustes de control requeridos por ONSPEC.
3. ONVIEW es el módulo que analiza los datos históricos. Permite al operador verificar el comportamiento del sistema, a través de gráficos que presentan la historia de las sucesivas lecturas.

Estos 3 elementos trabajan juntos y están interrelacionados. Los tres juntos son el corazón del sistema de control ONSPEC.

Los datos pueden provenir de una gran variedad de fuentes, y por lo tanto deberán ser representadas en forma apropiada: números reales, enteros o digitales. Todos los datos son almacenados en una Base de Datos central, junto con un indicador de estado que indica la calidad y condición del dato (un punto). Los valores dentro de las tablas de ONSPEC pueden ser modificados por las interfaces de entrada/salida, por el programa de control, por programas y funciones definidas por el usuario, y por comandos ingresados por el operador. Pueden definirse *tags* para identificar una entrada en la tabla con un nombre representativo. Esto es fundamental para la recolección de datos históricos, y pueden ser usados en displays y reportes.

La Interfaz de Entrada/Salida es responsable de transmitir información entre ONSPEC y los dispositivos de entrada/salida. Las comunicaciones desde y hacia los dispositivos de E/S, se realizan normalmente vía RS-232, aunque también existen algunos dispositivos que tienen salida paralela o DMA. Todas las interfaces de E/S, trabajan de la misma manera. El usuario define un archivo de configuración, en el cual se especifica la frecuencia de muestreo, la ubicación del dispositivo, la ubicación del punto, y la conversión a realizar. La comunicación con ONSPEC se realiza a través de colas a memoria, y también usando acceso directo a memoria. Muchos drivers utilizan la técnica de First-In First-Out (FIFO), para minimizar la cantidad de datos a procesar por el administrador de la cola.

ONSPEC provee una serie de 'moldes' para construir interfaces de entrada/salida entre la computadora y los dispositivos (Código Fuente). Cada molde fue escrito para un dispositivo de hardware específico y diseñado para sacar ventaja de las características del hardware, de forma tal que solo sean necesarios cambios mínimos para soportar otros protocolos. Se proveen mas de 100 moldes para distintos dispositivos de entrada/salida. Estos moldes son los que permiten la construcción de módulos de adquisición de datos, que luego serán manipulados y procesados por ONSPEC.

Las últimas versiones de ONSPEC incluyen una librería de objetos basada en el estándar ISA. El usuario puede agregar objetos a esa librería o puede crear librerías propias con nuevos objetos. Posee una interfaz gráfica con el usuario que puede ser configurada por él, para adaptarla a sus necesidades Por ejemplo las teclas de función pueden ubicarse en cualquier lugar del display (o no ser mostradas), el soporte de bit-maps tiene funciones para adaptarlos a los requerimientos del usuario.

Otra característica importante es que posee facilidades para el manejo de alarmas, usadas para monitorear los procesos. Cuando ocurre un evento crítico ocurre, las rutinas de servicio de la alarma generan una señal. Esta señal puede ser un mensaje, una bocina, ejecutar un proceso de control, o ejecutar procedimientos escritos por el usuario. Las alarmas se monitorean, según la frecuencia definida por el usuario.

Las alarmas se presentan en pantalla para permitir al operador monitorear los cambios en la planta. A su vez todos los cambios producidos se almacenan en un archivo histórico así como los datos que representan la planta se guardan en una base de datos central. Estos datos (tanto actuales como históricos), pueden ser visualizados a distintos niveles de detalle, ver en detalle los datos de un período en particular, usando la planilla de cálculo incluida en el paquete, o pueden ser procesados para generar algún tipo de reporte (por ejemplo reporte de tendencias históricas). La planilla de cálculo además permite manipular tanto los datos históricos como los datos de tiempo real.

Onview es una herramienta de ONSPEC que permite hacer presentaciones gráficas, monitoreo en tiempo real, y el análisis de datos históricos, producido por el software de control de ONSPEC. ONVIEW tiene acceso a los datos tiempo real de ONSPEC, así como también a los archivos históricos. La planta puede ser monitoreada y analizada a través de planillas de cálculo, de las tablas históricas, y gráficos de tendencias.

La opción de gráficos de tendencias, presenta gráficos interactivos de alta resolución, tanto para los datos tiempo real, como para los datos históricos. Los gráficos con datos tiempo real, se actualizan a medida que se leen nuevos valores del mundo real. Cuando se opera con datos históricos, el usuario puede seleccionar la porción de datos que se desea examinar, puede hacerlo con el detalle que desee, es decir, ver una porción de los datos tan en detalle como sea posible, o ver un esquema general, en un marco de tiempo mayor.

Onview, también ofrece todas las facilidades de una planilla de cálculos, como ser: cargar una planilla desde disco, grabarla, recalcular automáticamente luego de ingresar un nuevo valor, imprimir la planilla, eliminar la planilla actual y proteger celdas.

OCL es el lenguaje de programación de Onspec, de fácil uso que permite disparar acciones como ventanas de diálogo, otros displays y programas, según alguna cadena de eventos predeterminadas. OCL puede disparar programas REXX, C++ y Smalltalk.
La BD de ONSPEC así como los archivos pueden ser compartidos entre los distintos puestos de trabajo de ONSPEC. Provee soporte para comunicaciones con sistemas disímiles, como AS/400, Windows, Unix y VMS. Además la información dentro de ONSPEC, puede ser integrada con sistemas MRPII, y con bases de datos SQL distribuidas, como DB2, Oracle, Sybase, Paradox.

Posee 3 conjuntos de comandos distintos: para el operador, el supervisor, y el ingeniero. Cada conjunto posee los comandos acordes a las necesidades/nivel de seguridad de cada usuario. Por ejemplo, el operador tendrá acceso a un reducido conjunto de comandos para mantener el sistema ONSPEC. El supervisor, tendrá disponible un conjunto de comandos que no puede ser accedido por el operador. El ingeniero, tendrá acceso a un conjunto mas amplio de comandos que le permita construir y mantener la base de datos ONSPEC.

Existen algunas funciones disponibles que pueden ejecutarse automáticamente. Estas incluyen presentación automática de pantallas, planificación de eventos externos y iniciación automática del sistema.

- *Presentación automática de pantallas.* Pueden seleccionarse hasta 7 pantallas para ser presentadas a una hora seleccionada, las variables que deben completarse son :

  1. La pantalla a presentar
  2. La hora en que se desea hacer el log-in de la pantalla
  3. El incremento en horas, para el próximo log-in de la pantalla

- *Planificación de eventos externos.* Se pueden ejecutar una serie de eventos y re ejecutarlos, según la planificación definida por el usuario. Un evento, pude ser un comando un programa, o una serie de comandos y programas que se deben ejecutar.

- *Iniciación automática del sistema.* El sistema puede ser seteado, de manera tal que el programa apropiado se ejecute automáticamente, cada vez que la computadora se encienda.

El usuario tiene acceso a los archivos con el código fuente de ONSPEC. Esto le permite modificarlo para adaptarlo a sus necesidades. Si bien esto puede ser una gran ventaja, no es aconsejable hacer un gran número de modificaciones, ya que deberán repetirse, cada vez que se reciba una nueva versión de ONSPEC. Es responsabilidad del usuario documentar todos los cambios que se realizan.

La configuración de hardware necesaria para instalar ONSPEC incluye una computadora personal IBA AT o equivalente con un procesador 286 o 386 y 640K de memoria RAM (las versiones para 386 requieren 2 megabyte de memoria), una unidad de disco removible de 1.2 MB, un disco duro de 30 MB (lo recomendable es 40 MB), 1 o 2 ports serie, para los drivers o terminales remotas, 1 o 2 ports paralelos, 1 para informes y el otro para alarmas, palqueta gráfica color EGA, monitor EGA, Coprocesador matemático e impresora.

1.3.4.3 Otros SCADA

Otros SCADA conocidos en el mercado son:

LabVIEW. Este SCADA entra dentro de la categoría de instrumentación virtual [Kap95]. Brinda al usuario las herramientas necesarias para crear interfaces gráficas elaboradas. El software para drivers sobre el que está construido LabVIEW es estándar, y permite comunicación con instrumentos RS-232, GPIB y VXI, así como también con placas de adquisición de datos. Estos drivers permite que usuarios

con poco conocimiento de programación puedan programar interfaces con instrumentos de adquisición y control de los datos. Posee además una librería para análisis de los datos. Esta librería posee desde simple rutinas para análisis estadístico, hasta complejas rutinas para el proceso de señales digitales. LabVIEW se encuentra disponible para Macintosh, Windows y Sun.

PROMACE. Es un SCADA con interfaces con sistemas de manufactura e información gerencial. Recibe datos del mundo real los analiza y brinda información al usuario para la corrección de tendencias, todo esto en tiempo real. También almacena datos históricos para posterior análisis. Todos los datos se almacenan en una base de datos relacional. Posee un módulo de adquisición de datos y control de aplicaciones, que mantiene una imagen del proceso del mundo real. Permite que el usuario configure sus mímicos, pantallas, reportes y alarmas. El paquete incluye además un lenguaje de 4ta generación PCL (Process Control Languaje), que permite al usuario escribir secuencias de control simples o complejas, accediendo a la imagen del proceso y de esta forma al proceso mismo.

ObjectControl/System (OC/S). Provee un ambiente Orientado a Objetos y Cliente/Servidor para el desarrollo de sistemas SCADA. La Base de Datos Central es Orientada a Objetos (no es relacional). Posee una completa interfaz gráfica (OOGUI) y acceso a datos de tiempo real e históricos, los cuales pueden presentarse en planillas de cálculo EXCEL o usarse en aplicaciones Visual Basic. Como la aplicación es orientada a objetos esto permite que sea mas fácil de mantener y de extender. Esto se debe a que cada objeto posee una parte pública y otra privada. La parte pública es la interfaz con el sistema, la parte privada está oculta. Una vez que se define la parte pública del objeto, este puede ser modificado, pueden agregarse nuevas subclases, o nuevos métodos sin introducir cambios al resto del sistema. Posee un módulo de adquisición de datos compatible con una amplia gama de drivers. OC/S corre en ambientes Windows NT.

En el siguiente cuadro se resumen las principales características de software SCADA que se encuentran disponibles [Kap95].

Nombre del Producto	Sistema Operativo	Clase y Descripción	Comentarios y Características
Genie 2.0	Windows 3.1	(O) Incluye un lenguaje de programación similar a Visual Basic	Soporta DDE y conexiones para LANs
OMNI 4.0	QNX 4.2	(O) Paquete para Supervisión, Control y Adquisición de Datos	Incluye software para visualización de datos e interfaces con BD corporativas
AxoData 1.2	Macintosh 6.0.2	(E) Adquisición de datos y exposición de la información en forma rápida.	Es de uso general, pero fue hecho pensado en electrofisiología
pClamp 6.0	DOS 5.0	(E) Adquisición de datos, exposición y análisis de la información en forma rápida	Pensado para electrofisiología y electroquímica
TestPoint	Windows 3.1	(O) Adquisición de Datos, análisis del control de instrumentos y presentación para IEEE-488, RS-232	Herramienta fácil de usar para el desarrollo de aplicaciones de adquisición de datos para Windows
Universal Library	DOS, Windows	(I) Contiene drivers para todas las tarjetas de adquisición de datos	Soporta cualquier lenguaje bajo DOS o Windows
DasyLab	Windows 3.1	(V) software basado en iconos para adquisición de datos y control	E/S analógica y digital para gran número de tarjetas, IEEE-488 y RS-232. Adquisición de datos a muy alta velocidad
DSP-EZ	Windows 3.1	(A) Usa Visual Basic GUI	El lenguaje Basic de DSP ofrece un acceso fácil y rápido a los datos.
DSP Lab	DOS 5.0, Windows 3.1	(I) Kit completo para programación en C para DSP.	Incluye rutinas matemáticas y lógicas. Incluye un compilador C.
DT VEE	Windows 3.1	(V) Paquete completo para la programación de visual de software para adquisición y análisis de datos	Ahorra tiempo de desarrollo. Interfaz GUI que permite mostrar los datos y analizarlos sin necesidad de programar ningún módulo
VB-EZ	Windows 3.1	(A) Realiza la adquisición de datos y análisis y muestra los datos por pantalla en tiempo real en Visual Basic	Incluye ejemplos de programas que realizan e/s analógica y digital.
WinDaq/200	Windows 3.1	(E,I)Soft. de adquisición y presentación en pantalla de datos analógicos	No se requiere programación

Nombre del Producto	Sistema Operativo	Clase y Descripción	Comentarios y Características
PC-411	DOS, Windows 3.1	(V,I,F)Software especializado para las placas analógicas de E/S Datel´s de uso general PC-411 y PC-412	Incluye soporte para instrumentos virtuales, LabView, Dos y Windows. El código fuente esta disponible en su totalidad
PC-414	DOS, Windows 3.1	(V,I,F) Software para soporte de placas Datel para E/S analógica PC-414, de alta performance	Disponible el código fuente en C y visual C; posee una librería de módulos LabView
PC-430	DOS Windows 3.1	(I,V,F) Soporte para las placas analógicas de E/S de Datel PC-430 de alta performance, para DOS, Windows, LabView	Incluye librerías para adquisición de datos para alta performance
Instatrend 5.0	DOS, Windows	(E) Software para adquisición de datos y control de procesos para aplicaciones de laboratorio	Incluye software para alarmas, cálculos, PID, PLC e interfaz para voz
DADiSP/LT	DOS 5.0	(A)Provee control para el hardware adquisición de datos	Permite a los usuarios especificar los parámetros A/D, recolectar datos de la placa de adquisición de datos
Hydra Logger para Wiindows	Windows 3.1	(O) Paquete para adquisición de datos	No se limita a una aplicación específica
SuperScope II	Macintosh	(V) Instrumentación virtual completo	panel frontal de diseño flexible, análisis on-line
Snap Master	Windows 3.1	(A,V)software para adquisición de datos, muestreo en pantalla y almacenamiento basados en PC	No necesita programación. Adquiere datos de múltiples E/S al mismo tiempo
HP VEE 3.0	Windows, Unix	(V) Lenguaje de programación gráfico para el testeo y medición de aplicaciones	El usuario crea como el instrumento se deberá ver y controlar
HP3412A BenchLinkMeter	Windows 3.1	(E) Software Específico	Captura datos, los muestra por pantalla, los analiza y los imprime. Los datos capturados pueden ser usados en otras aplicaciones Window para posterior análisis o documentación
AMPS	Windows 3.1	(V) instrumentación virtual, que transforma la PC en un completo conjunto de instrumentos	Incluye diseño de filtros, analizador de espectros, 2 osciloscopios y almacenamiento digital
Genesis para Windows	Windows 3.1	(O,A,V)SCADA, MMI, permite ver los datos en tiempo real e históricos en Excel,; PID control	Incluye facilidades para operación en red, DDE, OLE 2.0 para interfaz con el operador
Pragon TNT	DOS, OS/2, Windows NT	(O)Soft. para automatización gráfica. Puede ser adaptado para cualquier requerimiento de una aplicación	intuitivo software de automatización, para monitoreo de procesos, control y manejo de información
Visual Designer 2.3	Windows 3.1	(O)Generador de aplicaciones para crear aplicaciones a medida sin programar	Soporta gran variedad de hardware de adquisición de datos, instrumentos y dispositivos seriales
VisualLAb	Windows 3.x	(A)Tiene extensiones Visual basic para desarrollo de aplicaciones para adquisición de datos	Software específico para adquisición de datos para windows
VTX	Windows	(A)Conj. de rutinas de control en Visual basic para el rápido desarrollo de aplicaciones para adquisición de dato	Hay disponibles módulos para adquisición de datos, análisis y gráficos, que ayudan a crear aplicaciones específicas
Reality	DOS	(E)Soporta varios canales y está basado en el concepto de sistemas abiertos	Soporte eficientemente desde pequeños a grandes sistemas, y puede ser usado con una o mas estaciones de trabajo Unix
Labtech Control	DOS 2.0 (o mayores), Windows 3.1	(V) Software que contiene gráficos para optimizar la visualización de datos de tiempo real	Incluye procesamiento de alarmas, PID control.
Collect, Collect/W	DOS 3.1, Windows 3.1	(A,O) Software que soporta múltiples canales RS-232, para recolección de datos de dispositivos seriales	Incluye una planilla de cálculo para datos de tiempo real.
LabStation 3.0	Windows 3.1	(O) comunicación vía RS-232 con instrumentos analíticos.	Instrumentos que pueden ejecutar stand-alone o agregarse a aplicaciones windows
Power -Assist Factory	MS-DOS	(O) Software para PLC	Incluye monitoreo de alarmas, análisis estadístico y mediciones en tiempo real

Nombre del Producto	Sistema Operativo	Clase y Descripción	Comentarios y Características
Origin DAQ	Windows 3.1	Incluye módulo para adquisición de datos en un paquete para análisis de datos y gráficos	Soporta tarjetas para adquisición de datos RS-232
DAPview, DAPwindows	DOS, OS/2, Windows	(O) software de PC para controlar placas de adquisición de datos, interfaz con sistema en tiempo real en PC	Cada placa DAP tiene su propia inteligencia: un coprocesador que corre aplicaciones bajo su propio SO, esto lo libera de las demoras impuestas por Windows, u otros programas que corren en la PC
DSPworks	Unix, Window 3.x	(T) Adquisición de datos para placas DSP	Soporta varios tipos de operaciones DSP
NI-DAQ	Es un SO que se ofrece con el hardware par adquisición de datos para DOS, Windows, Windows NT, Macintosh y Solaris	(I) Software para adquisición de datos	Es un SO operativo que elimina la programación de bajo nivel e integra el hardware con las aplicaciones. Sus rutinas pueden ser llamadas desde programas Visual Basic, Borland C++ y Microsoft C++
SpectrumWare	Se ofrece con placas para adquisición de señales para Macintosh, Windows y Sun	(E) Software para análisis de señales dinámico	GUI, fácil de usar, que simplifica la tarea de medir frecuencia de respuesta, amplitud de espectros, coherencia y distorsión
Collect	Windows 3.1	(I) Recolecta datos de instrumentos RS-232	Soporta la mayoría de los instrumentos para RS-232
Omega Trend	DOS 3.1	(T) Provee almacenamiento de datos	Fácil de usar rápido, exposición en pantalla en tiempo real
Spectra Plus 3.0	Windows 3.1	(T) Adquisición de datos en tiempo real, análisis de espectros usando tarjetas de sonido multimedia	Incluye facilidades para generador de señales
DATS+ V.2.1	DOS 5.0, Unix/Motif Windows 3.1,N	(I) Fácil interfaz con el usuario para hardware de adquisición de datos	
Camile TG	DOS 5.0, Windows	Software para adquisición de datos y control	Sistema completo que incluye tendencias, gráficos animados, procesamiento en red, DDE, SFC, CFC
Driver Link 2.0	Windows 3.x	(I) Software DLL para el desarrollo de aplicaciones de adquisición de datos en C, C++ y Pascal	Soporta placas de conversión A/D de 8 proveedores. Soporta WIN 32s, que es una interfaz de programación hecha por Microsoft, para código de 32 bits.
Driver Link / VB 2.0	Windows 3.x	(A) Software de control embebido en para Visual Basic, para el desarrollo de aplicaciones para adquisición de datos particulares	Soporta placas de conversión AD de 8 proveedores
N!Power	Unix, VMS	(T,I) Orientado a Objetos, de la familia de software X-Windows para el desarrollo de aplicaciones para adquisición de datos y análisis	Programación visual, lenguaje simbólico de comandos
Workbench	DOS 5.0, Macintosh, Windows 3.1	(O) Software configurable, escrito en C, permite agregar iconos programados por el usuarios	Incluye DDE, RS-232, IEEE 488, estadística, gráficos X-Y; help sensible al contexto, y un tutorial
Software Wedge	DOS, Windows 3.x	(A) Adquisición de datos en tiempo real para cualquier programa de PC	Recolecta datos de cualquier instrumento conectado a la RS-232, o port serial. Posee características de avanzada.
Ultrasoft 2.4	Unix	(I,F) Contiene software para hacer llamadas a rutinas de adquisición de datos en C. Contiene código fuente que puede ser compilado en cualquier programa C	Soporta placas de conversión AD/DA.
VisSim	Windows 3.1, Windows NT	(O) Aplicación con drivers de adquisición de datos	Incluye capacidad de simulación y control, MMI, facilidades para análisis, E/S analógica y digital, soporta placas de los proveedores mas conocidos.

(A) = Facilidades Agregadas
(E) = Software Específico
(F) = Código Fuente
(I) = Interfaz con un Lenguaje de Programación
(V) = Instrumentación Virtual
(O) = Otros

1.4 Diseño de sistemas en Tiempo Real usando MASCOT

Para el diseño de sistemas de tiempo real es necesario considerar requerimientos adicionales, no contemplados en las metodologías de diseño para sistemas de gestión o comerciales. Esto se debe a que los sistemas en tiempo real, consisten en varios procesos o tareas ejecutándose concurrentemente, en forma asíncrona y a diferentes velocidades, pero algunas veces estas tareas necesitan sincronizarse para poder cumplir con los requerimientos funcionales del sistema.

En reglas generales, el ciclo de vida de un sistema en tiempo real no difiere demasiado del ciclo de vida de cualquier otro tipo de sistema. La diferencia básica es que los sistemas en tiempo real hacen énfasis en la descomposición del sistema en tareas concurrentes, en la definición de las interfaces entre estas tareas y en la integración de las mismas. El ciclo de vida de un sistema en tiempo real consta de las siguientes etapas [Gom86]:

- *Análisis de Requerimientos y especificación:* los requerimientos del usuario son analizados y se especifica el sistema para que cumpla con estos requerimientos
- *Diseño del Sistema:* el sistema es estructurado en tareas (procesos concurrentes), y se definen las interfaces entre tareas
- *Diseño de la Tarea:* cada tarea se estructura en módulos. Se definen las interfaces entre los módulos de una tarea
- *Construcción:* diseño detallado, codificación y prueba unitaria de cada módulo
- *Integración de Tareas:* se integran los módulos que conforman una tarea, y se prueba cada tarea. Cuando todas las tareas funcionan según los requerimientos, se integran todas las tareas en un sistema
- *Prueba del Sistema:* se prueba todo el sistema para verificar que se cumplan los requerimientos funcionales.
- *Prueba de aceptación:* esta prueba es realizada por el usuario.

Se analizaron las siguientes metodologías de diseño: DARTS (Design Approach for Real Time Systems) [Gom84], Desarrollo estructurado para sistemas en tiempo real [War86] y Mascot (Modular approach to software construction, operation and test) [Jak75]. Para realizar la selección se tuvo en consideración que se cumpliera con los siguientes requerimientos [Gom84], que son esenciales en una adecuada metodología de diseño de sistemas en tiempo real:

- *Diseño Orientado al flujo de datos.* Usar la técnica de Flujo de Datos para el diseño de software es apropiado para los sistemas en tiempo real, ya que en estos, puede considerarse que los datos fluyen de la entrada a la salida, y en el medio son transformados por tareas o procesos. En general los sistemas en tiempo real son centrados en transacciones, es decir el flujo consiste en información de control o datos, que es pasada a un proceso a partir del cual se inicia una acción o secuencia de acciones, basada en los datos de entrada. La notación básica de esta técnica es (ver Figura 10) una burbuja que representa la transformación de los datos y el flujo de datos que se representa a través de flechas.

Figura 10
Diagrama de Flujo de datos - Notación básica

- *Comunicación y sincronización de tareas.* En los sistemas de tiempo real es esencial poder sincronizar y comunicar las tareas o proceso que lo involucran, dado que no existe un flujo predeterminado de ejecución de las tareas, sino que son activadas según diversas circunstancias del entorno.

  En cuanto a la *sincronización* de tareas en tiempo real, existen básicamente dos tipos : exclusión mutua y estimulación cruzada. La exclusión mutua es requerida cuando los datos pueden ser accedidos concurrentemente por dos o mas procesos. Una forma de implementación es a través de semáforos binarios. La estimulación cruzada ocurre cuando una tarea está esperando una señal de otra tarea para poder continuar. Esto último también puede implementarse usando semáforos binarios.

  La *comunicación de tareas,* ocurre cuando una tarea *productora,* necesita pasar información a una tarea *consumidora.* La forma mas común de implementar esto es a través de *mensajes.*

  La comunicación puede ser fuertemente acoplada, donde cada vez que el productor manda un mensaje se quedará esperando la respuesta del consumidor, o débilmente acoplada donde el consumidor deja sus mensajes en una cola y sin esperar respuesta del consumidor continúa su rutina. Por otro lado, cada vez que el consumidor termina de procesar una tarea busca en la cola el próximo mensaje a procesar

- *Ocultamiento de Información o Encapsulamiento.* El objetivo es dividir el sistema en módulos de manera tal que cada componente encapsule u oculte una decisión de diseño. La interfaz de ese módulo es definida para que no pueda adivinarse el contenido del mismo. La ventaja de este método es que los módulos son autocontenidos y por lo tanto el sistema es mas fácil de modificar y de mantener. En otras palabras se reduce el acoplamiento entre módulos. Si bien el ocultamiento de información no es un requerimiento esencial para sistemas en tiempo real, es un concepto que permite gran flexibilidad en la definición de interfaces y dado que estos sistemas tienen una gran variedad de interfaces con la planta, entonces es recomendable diseñarlas usando este concepto, lo que permite aislar la interfaz de la real implementación.

- *Dependencia de estados en el proceso de transacciones.* Muchos de los sistemas en tiempo real son orientados a transacciones, pero además en muchos casos la acción a tomar no solo depende de los datos de entrada sino también del estado actual del sistema, es decir de lo que pasó anteriormente. Una forma de implementar esto sería a través de un diagrama de transición de estados que permite saber, dado el actual estado de la planta cuales son todas las acciones válidas a seguir.

Se decidió usar Mascot como herramienta de diseño por ser esta una herramienta simple, que usa la técnica de descomposición de actividades basada en el flujo de datos, con algunos agregados para representar control, interfaces entre las distintas tareas y la comunicación y sincronización de las mismas. También usa el concepto de ocultamiento de información para acceder a los datos de los canales y 'pools' . El acceso a estos se hace a través de llamadas a rutinas. Todas estas características hacen que la metodología pueda integrarse con nuestra herramienta [Wai94], permitiendo disponer de un entorno de desarrollo completo, que permite construir aplicaciones de supervisión complejas a bajo costo y alta seguridad. De hecho pudimos comprobar esto en nuestra experiencia, lo que será desarrollado más adelante.

1.4.1 MASCOT

En el diseño preliminar, Mascot se basa en la construcción de una máquina virtual para resolver un problema determinado. En las etapas posteriores del diseño, se implementa esta máquina virtual en

una o mas computadoras, ya que el método de diseño no hace ninguna restricción respecto de la cantidad de procesadores a utilizar.

Veremos a continuación una serie de entidades abstractas que se usan para diseñar los sistemas

1.4.1.1 Actividades

Cada actividad (proceso de transformación) que se deba ejecutar, será representada por una burbuja o círculo. Varias actividades podrán combinarse para formar una única tarea o proceso. Las actividades que conformen una tarea deberán poder comunicarse entre si y sincronizar sus acciones.

1.4.1.2 Comunicación

La comunicación entre actividades puede dividirse en tres tipos:

- Transferencia directa del flujo de información entre dos actividades (Figura 11): la información viaja de una burbuja a la otra. Las actividades se comunican a través del flujo de información que se representa a por una flecha. Esta fecha indica desde y hacia donde fluye la información.

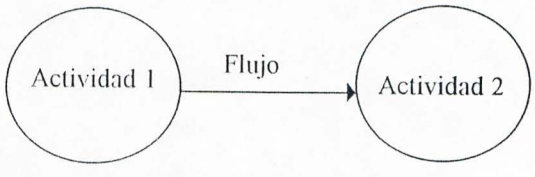


Figura 11
Transferencia Directa

- Compartir información entre varias actividades (Figura 12): otra forma de comunicación es un usar un repositorio de datos que puede ser accedido por mas de una tarea para lectura y/o escritura. Las flechas muestran el flujo de datos entre las actividades y el repositorio.

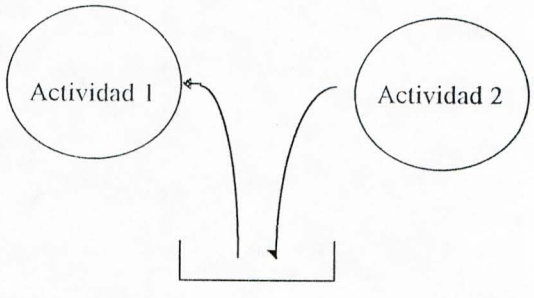


Figura 12
Información Compartida

- Señales de sincronización (Figura 13): estas señales se utilizan con el propósito de sincronizar actividades, sin ser necesario la transferencia de información entre ellas. Esta señal puede significar que la actividad receptora espere la ocurrencia de un evento, o que la actividad emisora señale la ocurrencia de un evento que active a la receptora.

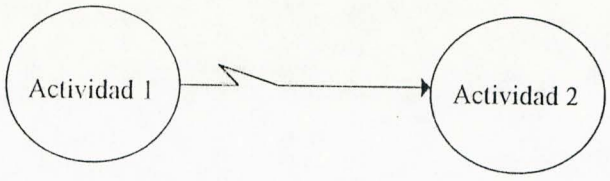


Figura 13
Señal de Sincronización

1.4.1.3 Canales

Un concepto útil para describir la comunicación directa entre actividades son los *canales*. Este es un concepto similar al de *pipe*. Un canal, además de conectar dos actividades, debe tener cierto tipo de almacenamiento asociado, de manera tal que pueda pasar a través del canal mas de un ítem de información a la vez. Los datos normalmente se ordenan de manera tal que el primero en pasar a través del canal sea el primero en ser procesado en el otro extremo del canal. En la etapa de diseño no es necesario determinar cómo se va a implementar el canal. Un canal se representa como vemos en la Figura 14.

Figura 14
Representación de un canal

1.4.1.4 Repositorios

Es un conjunto de información disponible para lectura y/o escritura por parte de las actividades del sistema. La operación de lectura no modifica la información en el repositorio. En el diseño no debe considerarse cómo van a operar los repositorios. Sin embargo debe asumirse que en cualquier implementación habrá mecanismos para proteger los datos de posibles corrupciones, y que garanticen un acceso ordenado a los datos. Vemos la notación para los repositorios en la Figura 15.

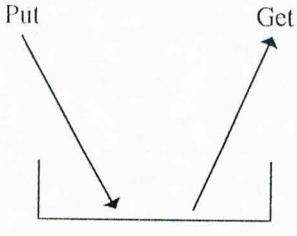


Figura 15
Representación de Repositorios

Los dispositivos físicos (archivos, pantallas), incluyendo los dispositivos de la planta que se desea controlar, pueden representarse como repositorios.

1.4.1.5 Sincronización

Las actividades necesitan poder *parar, comenzar,* y *retrasar* su ejecución o la de otra tarea. También necesitan sincronizarse con otra tarea para saber cuando esta está lista para recibir o mandar datos.

Una técnica muy usada para la sincronización de tareas, es utilizar dos procedimientos: *Wait* (evento) y *Signal* (evento), con las propiedades que se definen a continuación.

1. WAIT (evento) La tarea suspende la actividad tan pronto como la operación Wait es ejecutada, y continua suspendida hasta tanto el *evento* no ocurra.
2. SIGNAL (evento) La operación Signal es la que 'avisa' a la tarea suspendida que el evento ocurrió y por lo tanto este reinicia su ejecución.

En sistemas en tiempo real que realizan el control de una planta, el SIGNAL sobre un evento se realiza a través de interrupciones externas. Estas pueden considerarse como señales de sincronización. A su vez una interrupción externa, que requiera alguna acción para corregir el funcionamiento del entorno, se traduciría en que la actividad sobre la que se hizo el Signal, estará en Wait, es decir a la espera alguna acción para corregir el mal funcionamiento.

1.5 Selección de herramientas

La elección del software de desarrollo y del sistema operativo no son tareas simples. Hoy en día se ofrecen una gran cantidad de productos, pero en el momento de la elección existen diversos factores determinantes. En esta sección estudiaremos las herramientas elegidas para el desarrollo, justificando su elección.

1.5.1 Selección del Sistema Operativo

Para la elección del SO bajo el cual desarrollar la herramienta para la construcción de supervisores para sistemas en tiempo real, se consideraron varios de los SO más difundidos que se encontraban disponibles en el momento de tomar la decisión. A continuación enunciaremos los pros y contras de los SO analizados[2].

DOS: Este sistema operativo es el mas difundido en todo el mundo y por esto posee un gran mercado. Requiere una configuración de hardware mínima, es muy fácil de usar y posee estándares uniformes. Existe una gran variedad de herramientas de desarrollo para este SO, y una gran cantidad de drivers de periféricos para todos los dispositivos de hardware. Posee además, un ambiente de desarrollo estable. Pero tiene grandes desventajas ya que no es multitarea y no posee facilidades de operación en red. Otra importante desventaja es el límite de 640K de acceso a memoria. No posee entorno gráfico, es orientado a comandos y no hace buen uso de la capacidad de los nuevos procesadores (ej.: bus de dirección de 32 bits, grandes cantidades de memoria RAM).

Windows: Si bien este no es un sistema operativo real, ya que corre sobre DOS, junto con este último conforma una plataforma de desarrollo y ejecución muy utilizado en la actualidad. Esto de debe a que es muy fácil de usar y a que posee un entorno gráfico (GUI); estas condiciones orientan su uso a usuarios sin experiencia en computación. Una desventaja importante de este entorno es que no es multitarea real, su entorno de desarrollo no es estable y no es muy rápido.

---

[2] Este análisis y sus conclusiones fueron realizados en Marzo de 1994, fecha en la que comenzó la construcción de la herramienta.

UNIX: Este SO es multi usuario y multitarea, está muy difundido en el ámbito académico y en producción. Existe gran variedad de herramientas que facilitan el desarrollo de aplicaciones. Pero aquí el problema, es que en muchas versiones la política del algoritmo de planificación es round-robin con prioridades y sin desalojo del núcleo. Esto lo consideramos una desventaja muy importante para el desarrollo sistemas en tiempo real, ya que si se está ejecutando una tarea y llega otra de muy alta prioridad, (por ejemplo una rutina de atención a una alarma), deberá interrumpirse la tarea que se está ejecutando para procesar la de mayor prioridad. Otros aspectos desfavorables son: la configuración de hardware requerida es costosa, es orientado a comandos, no posee entorno gráfico y su instalación no está orientada a usuarios finales[3].

MINIX: Este sistema operativo es un clon de Unix muy utilizado en el ámbito académico. Requiere una configuración de hardware mínima, los servicios que posee son muy simples, el código fuente se encuentra disponible por completo para ser modificado, y además posee facilidades para operación en red.

Sin embargo no nos pareció adecuado ya que el planificador de tareas de este SO utiliza la técnica de Round Robin y esto no es adecuado para el desarrollo de sistemas en tiempo real donde se requiere una planificación de tareas según las prioridades de las mismas. Adicionalmente encontramos los siguientes problemas: no se proveen herramientas de debbugging para la programación, no provee mecanismos de memoria compartida, el kernel permite ejecutar un máximo de 32 tareas en forma concurrente, no pueden considerarse intervalos de planificación menores a 1 segundo, no posee entorno gráfico entre otros.

QNX: Este sistema operativo tiene el atractivo de haber sido diseñado específicamente para aplicaciones en tiempo real. Es multitarea y multiusuario, y es distribuido, es decir está dividido en módulos que pueden ubicarse en distintos puestos de trabajo, siendo esto transparente para el usuario final. Su arquitectura es modular, con un microkernel de tamaño reducido (10 K). Posee facilidades de operación en red y variedad de algoritmos de planificación como FIFO, round-robin, adaptativo y de prioridades. Otra característica importante es que no hace swapping de tareas por lo que trabaja con todos los procesos en memoria RAM. Esto acelera los tiempos de cambio de contexto. Esta característica es de gran importancia en sistemas en tiempo real, ya que tienen fuertes restricciones de tiempo.

Pero, a pesar de todo esto, QNX es un SO muy poco difundido dentro del país, y existen en el mercado muy pocos elementos de software desarrollados para este SO, y por ende, existen pocas herramientas de desarrollo. Por otra parte, si bien se dice que es un SO en tiempo real, sus algoritmos de planificación no aseguran que un proceso pueda ejecutarse dentro de un límite fijo de tiempo. Otra desventaja importante es el costo del hardware necesario para una configuración un poco mejor que la básica: para montar un entorno de desarrollo se requiere una configuración de hardware costosa. Por otro lado al no hacer swapping de tareas, se necesita una gran cantidad de memoria RAM para ejecutar una cantidad de procesos concurrentemente, lo que también incrementa el costo. Por último diremos que es orientado a comandos y que no provee mecanismos de memoria compartida

Windows-NT: En la actualidad este sistema operativo se está volviendo muy popular para la administración de redes locales (LAN). Es fácil de usar, de configurar y de mantener, a la vez está bien integrado con otras aplicaciones Windows, muy populares en el mercado. También el hardware necesario para un server NT es relativamente bajo, debido a que NT se basa en plataformas INTEL. Otra ventaja de este sistema operativo es que posee un nivel de rutinas que permiten independizar al SO del hardware (HAL, hardware abstraction level), lo que a su vez permite a Windows NT soportar una amplia variedad de diseños de hardware dentro de una familia de microprocesadores.

---

[3] En este análisis incluimos las principales versiones de Unix: Aix, Solaris, SCO, Linux, etc.

Una desventaja importante para este sistema operativo es la performance, y que no es escalable. En las últimas versiones, Micrisoft trató de mejorar la performance haciendo que las funciones primitivas del SO se comuniquen directamente con el hardware, pero esto provocó que el sistema sea menos estable.

OS/2: Por último hablaremos de este SO que es el que elegimos como base para nuestro desarrollo por tener un gran número de importantes características: es multitarea, hace buen uso del bus de 32 bits, posee un algoritmo de planificación por prioridades con remoción, con lo que se puede emular la gran mayoría de los algoritmos de planificación de tiempo real existentes. Por otro lado es una plataforma de SO por lo que en un ambiente OS/2 puede correrse cualquier proceso DOS ó Windows como otra independiente. Esto lo hace muy atractivo como elección porque como se dijo anteriormente DOS es un SO ampliamente difundido, por lo que el usuario final puede correr todas las aplicaciones a las que estaba acostumbrado y por lo tanto no restringe su uso a las aplicaciones OS/2 nativas. OS/2 posee facilidades de interoperabilidad e interconexión con otras plataformas; es orientado a usuarios sin experiencia con computadoras; posee un entorno gráfico orientado a objetos (OOGUI); trabaja en modo protegido, proveyendo un entorno de desarrollo muy estable. Finalmente existe un gran número de herramientas que facilitan su uso.

El mayor problema de OS/2 es que requiere una configuración de hardware relativamente costosa y no es sencillo de instalar, para montar un entorno de desarrollo práctico y por otro lado no corre en procesadores 8086. Pero en comparación con las otras opciones el costo es inferior, permitiendo que la herramienta pueda usarse en variedad de entornos en la industria y académicos.

1.5.1.1 Características de OS/2

Los servicios básicos de OS/2 pueden dividirse en 2 grupos que conforman los componente principales de este SO : Kernel y Presentation Manager. El Kernel provee los siguientes servicios :

- **Multitarea.** OS/2 provee un nivel de multitarea granular. La arquitectura del SO se basa en tres tipos de tareas que pueden ejecutarse: threads, procesos y sesiones.

  Un *thread* es la mínima unidad de concurrencia y de asignación de CPU en OS/2. El *proceso* provee el nivel de concurrencia que corresponde a un programa en ejecución. Consiste de uno a mas threads, lo que significa que OS/2 permite tener mas de un programa ejecutando concurrentemente y a su vez cada programa puede tener varios threads que estén corriendo al mismo tiempo. El nivel mas alto de concurrencia está dado por la *sesión* (o grupo de pantallas). Este nivel representa un grupo lógicamente separado de pantalla, teclado, mouse y el grupo de procesos asociados a estos recursos. Podría pensarse la sesión como una computadora virtual.

  OS/2 planifica el uso del procesador usando un algoritmo basado en prioridades con desalojo. Un thread puede ser asignado a una de las cuatro clases de prioridades, dentro de cada clase el planificador de tareas reconoce hasta 32 niveles de prioridades. Las clases de prioridades son :

  * *Críticas.* Para los threads que requieran atención inmediata. Esta clase es usada en aplicaciones de tiempo real.
  * *Alta.* Para aquellos threads que requieran un buen tiempo de respuesta, sin llegar a ser críticos
  * *Regular.* Para aquellos therads de ejecución normal.
  * *Ocioso.* Para threads de baja prioridad

  OS/2 combina las prioridades con la técnica de Round Robin para asegurarse que threads con igual prioridad tengan las mismas chances de ejecutar. Este elaborado algoritmo de planificación de tareas que usa OS/2 lo hace adecuado para desarrollar aplicaciones en tiempo real que tienen un tiempo de respuesta determinístico. El sistema garantiza que los threads críticos sean despachados

en 6 milisegundos, a partir de que están listos para ejecutarse. El tiempo máximo que una interrupción puede estar desactivada es 400 milisegundos. Finalmente OS/2 soporta un total de 4096 threads y procesos, lo que lo hace apropiado para desarrollar aplicaciones de Tiempo Real complejas.

- **Recursos Virtuales.** La base para soportar la concurrencia es manejar los recursos físicos de la computadora de forma tal que múltiples programas puedan ejecutar al mismo tiempo sin molestarse mutuamente a medida que van usando los recursos de la PC.

  OS/2 se ubica entre los programas y el hardware, y regula los accesos a memoria, discos, impresoras, pantallas, modems, etc. Las aplicaciones interactúan con memoria y dispositivos virtuales, los que a su vez OS/2 mapea a los dispositivos reales. OS/2 mantiene ordenado el acceso, cuando varios programas quieren acceder al mismo recurso.

- **Comunicación entre Procesos.** Como pueden ejecutar varios procesos concurrentemente, es necesario que se comuniquen entre sí para el intercambio de información . OS/2 provee esta facilidad a través de un conjunto de protocolos (IPC - InterProcess Communication protocols). Alguno de los protocolos provistos por OS/2 son:

  * *Pipes.* pueden ser de dos tipos: anónimos y con nombre. Los pipes anónimos son buffers de longitud fija, que pueden accederse como archivos de caracteres, a través de punteros de lectura y escritura. Estos pipes son en su mayoría usados por un proceso padre para comunicarse con sus descendientes. Los pipes con nombre, proveen una forma de comunicación entre procesos que no están relacionados. Para acceder al pipe los procesos lo hacen a través de los servicios de acceso a archivos de OS/2. Podemos asimilar estos servicios a canales de MASCOT.

  * *Colas.* Permite que información enviada por múltiples procesos sea leída por un único proceso. Este intercambio no tiene por que estar sincronizado. El proceso que recibe, puede acceder a la información de las siguientes maneras: FIFO, LIFO o por prioridad. Las colas en OS/2 contienen punteros a los elementos de la misma. Este servicio también puede asimilarse a canales MASCOT.

  * *Memoria Compartida.* Es otro protocolo de comunicación entre procesos. OS/2 provee la facilidad de crear objetos en memoria compartida. Cualquier proceso que conozca el nombre del objeto, puede accederlo. Los procesos deberán coordinar el acceso a este objeto en memoria compartida, a través del uso de semáforos. Este servicio implementa una forma de repositorios MASCOT.

- **Sincronización entre Procesos.** Es el mecanismo utilizado para que procesos o threads concurrentes no interfieran entre si, cuando tratan de acceder a recursos compartidos. La idea es acceder al recurso compartido en forma serial, usando un protocolo que todos estén de acuerdo en seguir. OS/2, como la mayoría de los SO posee semáforos para la sincronización de procesos.

  OS/2 tiene dos tipos de semáforos: privados y compartidos. Los semáforos privados se usan para sincronizar threads dentro del mismo proceso. Un proceso puede tener hasta 64 semáforos privados. Los semáforos compartidos están disponibles para todos los procesos en el sistema. Puede haber hasta un máximo de 64 semáforos compartidos.

- **Enlace Dinámico.** Permite a un programa tener acceso en tiempo de ejecución a funciones que no son parte de su código ejecutable. Estas funciones están empaquetadas en bibliotecas (DLL - Dynamic Link Libraries), que contienen código ejecutable pero que no pueden ser ejecutadas como aplicaciones. La aplicación puede cargar la DLL apropiada y ejecutar las funciones contenidas en ella, haciendo un enlace dinámico, es decir la aplicación se linkedita con las rutinas

de la librería en tiempo de ejecución en lugar de hacerlos en tiempo de compilación. El enlace dinámico permite a una aplicación cargar la DLL solo cuando necesita ejecutar una función de la biblioteca. Una vez que la DLL está cargada en memoria, puede ser compartida con otras aplicaciones, porque en un momento dado se mantiene solo una copia de la DLL en memoria.

- **Sistema de Archivos.** OS/2 tiene dos sistemas de archivos, File Alocation Table (FAT), y High Performance, File System. (HPFS). FAT es el sistema de archivos por defecto, y no necesita ser instalado. HPFS puede ser instalado durante la inicialización del sistema. OS/2 maneja archivos en disco y dispositivos de la misma manera. Por ejemplo, una aplicación usa la misma API (Application Programming Interface) para abrir y leer de un archivo en disco, que para abrir y leer un por serial.

  El sistema de archivo de OS/2 es compatible con el sistema de archivo de MS-DOS. Ambos sistemas de archivos representan una jerarquía de archivos que se encuentran en el disco. Cada disco a su vez puede dividirse en discos lógicos o particiones, donde cada uno tiene su propia jerarquía.

  Usando el archivo CONFIG.SYS de OS/2, puede especificarse el número de Kbytes, de la cache en RAM. El sistema usa el algoritmo LRU (Last Recently Used) para accederla. OS/2 provee facilidades para bloquear y compartir archivos, para soportar el acceso a los mismos en forma concurrente.

  El sistema de archivos HPFS permite el manejo de archivos de gran tamaño en forma rápida y consistente. Este sistema de archivos hace posible soportar grandes bases de datos en la PC.

- **Administración de Memoria.** OS/2 administra la memoria de forma tal que soporta aplicaciones DOS, Windows y OS/2. Todas las aplicaciones corren como procesos OS/2, sin importar su origen.

  Algunas de las características de este administrador de memoria son :

  * Provee a cada proceso de su propio espacio de direccionamiento virtual, el cual es de 512 MBytes para aplicaciones y de 4 GBytes para procesos privilegiados del sistema. OS/2 provee protección contra violaciones de memoria, y previene que programas que están ejecutando concurrentemente se invadan mutuamente sus espacios de direccionamiento.

  * Usa el paginado de los procesadores Intel 30386 (y superiores), para proveer un entorno de memoria virtual paginada por demanda. Que cada proceso pueda tener 512 MBytes de memoria virtual, permite ejecutar programas muchas veces mas grandes que la memoria física disponible en la computadora. La memoria es asignada en bloques de 4 KBytes. OS/2 crea un entorno de memoria virtual, dando a cada proceso su propio conjunto de tablas de páginas. Cuando un proceso está ejecutando y la página que necesita no está en memoria, OS/2 automáticamente manda al disco una de las páginas usando el algoritmo LRU, y hace lugar para la nueva página que fue requerida por el proceso.

  * Se provee una API (Application Program Interface) para crear objetos en memoria. También permite compartir los objetos en memoria con otros procesos. Estos objetos compartidos residen en un espacio de direccionamiento reservado, que comienza en la parte superior del espacio de direccionamiento del proceso y crece hacia abajo. En la
  * Figura 16, podemos ver el espacio de direccionamiento de OS/2. Como podemos ver el espacio de direccionamiento del sistema está por encima del espacio de direccionamiento del proceso. La memoria compartida crece desde arriba hacia abajo, mientras que la memoria privada desde abajo hacia arriba.

* OS/2 junto con el hardware del 80386, asegura que el código con menos privilegio permanezca dentro de los límites de su espacio de direccionamiento. Controla el acceso a instrucciones de entrada /salida y a interrupciones, usando el nivel de privilegio de entrada/salida del programa. El mecanismo de nivel de privilegio, previene a un programa de acceder cualquier parte del SO en forma incontrolada. OS/2 soporta 4 niveles de privilegio, los cuales se ordenan jerárquicamente desde el nivel 0 (máxima protección), hasta el nivel 3 (mínima protección). El corazón de OS/2 corre en nivel 0; el nivel 1 no es usado por OS/2; el nivel 2 por programas que directamente acceden a dispositivos de entrada/salida; el nivel 3 es donde corren los programas normales OS/2.

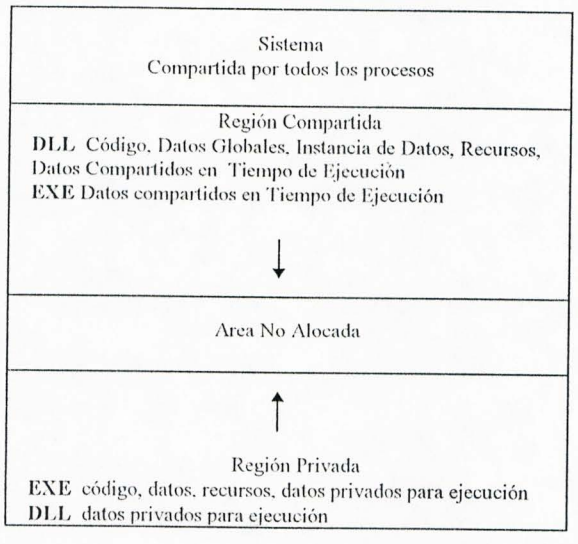


Figura 16
OS/2 - Espacio de direccionamiento de un proceso

- **Administrador de Excepciones.** En un ambiente multitarea, un error serio ocurrido en una aplicación, no debe afectar a otras aplicaciones que están corriendo al mismo tiempo. Errores inesperados, como por ejemplo violaciones de memoria, son llamadas excepciones. Cuando OS/2 detecta una excepción, usualmente finalizará la ejecución , a menos que la aplicación haya programado su propia rutina de atención de excepciones. Esta función permite a la aplicación corregir el error y continuar ejecutando. Estas funciones pueden programarse en algún lenguaje de alto nivel (como por ejemplo C).

Además de los servicios del kernel los servicios de presentación de OS/2, proveen una poderosa interfaz gráfica con el usuario (GUI - Ggraphical User Interfase). Estos servicios son llevados a cabo por el Administrador de los Servicios de Presentación. Estos servicios están escritos en C, y hacen uso de la ventajas de trabajar en un entorno de 32 bits, lo que se resume en un mejor tiempo de respuesta. También son orientados a objetos. Algunos de los servicios son:

- **Servicios de Programación**

  * Manejador de Ventanas: Conjunto de APIs para manejo de ventanas. Esto incluye crear, mover, cambiar el tamaño de ventanas y del contenido de las ventanas
  * Primitivas Gráficas: conjunto de primitivas que permite crear objetos gráficos, que pueden ser mostrados en pantalla, impresos o almacenados. Las primitivas tienen un conjunto de parámetros como color, tamaño, tipo de línea, sombreado, posición el la pantalla, etc.

* Manipulador de Objetos Gráficos: conjunto de APIs que permiten transformar objetos gráficos, como por ejemplo cambiar el tamaño, rotarlo y trasladar la posición.
* Espacios de presentación: permite asociar un objeto a un espacio virtual , de esta manera el objeto es independiente del dispositivo en el que se lo va a escribir.
* Bitmaps: OS/2 provee un conjunto de APIs para la manipulación de bitmaps.
* Clipboard : Permite el intercambio de datos entre distintas aplicaciones usando el clipboard
* Drag & Drop : Ofrece una serie de funciones para manipular los objetos directamente

- **Sistema de Modelado de Objetos.** OS/2 incluye una facilidad de programación orientada a objetos llamada Sistema de Modelado de Objetos (SMO). Esta facilidad permite crear librerías de clases en cualquier lenguaje que se elija. No es un lenguaje de programación, sino un sistema para definir y manipular librerías de clases. SMO se usa para definir clases orientadas a objetos y las relaciones entre ellas (como herencia, encapsulamiento y sobreescritura de métodos. Podemos decir que SMO es un entorno orientado a objetos para programas.

- **Facilidad para la Presentación de Información.** Es una herramienta muy útil, que puede ser usada para realizar las ayudas de una aplicación en forma sensible al contexto o crear hipertextos .

1.5.2 Selección del lenguaje de programación

Para seleccionar el lenguaje de programación, se tuvo en cuenta que este permita codificar a bajo nivel, para poder explotar al máximo las características del sistema operativo (por ejemplo para utilizar semáforos, mensajes, etc.), y programar funciones que accedan directamente al hardware (como por ejemplo las rutinas de manejo de ports). Al mismo tiempo buscamos que el lenguaje de programación utilizado tenga las facilidades de alto nivel necesarias como para permitirnos construir las interfaces en forma rápida y simple.

Entre todos los lenguajes disponibles para OS/2 que analizamos (Pascal, C, C++), encontramos que el C++, por un lado es un lenguaje de bajo nivel por poseer todas las características de bajo nivel del C, y por otro lado las clases del C++ brindan las facilidades de programación de alto nivel necesarias.

Otra ventaja de este lenguaje es que es orientado a objetos, es decir posee los siguientes mecanismos :

- *Abstracción de datos.* Significa que las características de un objeto se encuentran definidas y documentadas en la descripción de la clase.
- *Encapsulamiento.* El código y los datos se almacenan como una sola unidad. El encapsulamiento también permite el ocultamiento de la información ya sea total o parcial, ya que porciones del código o datos, pueden hacerse inaccesibles desde afuera de la unidad.
- *Modularidad.* El código se encuentra divido en funciones de manera tal que puede ser reusado.
- *Jerarquías.* Permite que el comportamiento de un objeto pueda ser redefinido (subclases), sin necesidad de recodificar el objeto padre (super-clase).Es decir la clase hija hereda las funciones y atributos de la clase padre. C++ permite que un objeto tenga mas de una super-clase. Esto se conoce como herencia múltiple.
- *Mensajes.* Su función es similar a la llamadas a funciones, es decir, los mensajes son requerimientos de realizar una operación determinada, por una instancia de un objeto.

Decidimos usar este paradigma de programación para mejorar el proceso de diseño de la aplicación. Las ventajas de esta técnica son :

- Las funciones son mas fáciles de mantener y de modificar.
- Permite modificar o cambiar la definición de alguna función de forma transparente para el usuario.
- Permite identificar y reusar componentes comunes.

- Permite usar un objeto de uso general para un uso mas específica (por ejemplo la clase TABLA se usó para definir una clase mas específica TABLA_INT, que representa a las tablas de valores enteros).
- Soporta creación y destrucción dinámica de objetos. Es decir en tiempo de ejecución, a medida que se identifica la necesidad de un nuevo objeto, se crea dinámicamente; no es necesario conocer de antemano el número máximo de los objetos que pueden ser necesarios en tiempo de ejecución
- El uso de clases permite aislar las funciones de mas bajo nivel y que por lo tanto están mas relacionadas con el SO usado. La ventaja de esto es que como están bien identificadas, en caso de querer cambiar el SO, estas funciones son fácil de reemplazar.

1.5.3 Otras herramientas utilizadas.

Para construir el Supervisor de ejemplo usando IGNATIUS, utilizamos una herramienta de programación visual, VISPRO C++ 1.0, que genera código C++, para el desarrollo de la interfaz con el operador. Este entorno combinado con nuestra herramienta permite disponer de un ambiente de desarrollo de aplicaciones SCADA, que minimiza el esfuerzo requerido para la codificación.
Utilizamos también VISPRO C++ para codificar las rutinas particulares del supervisor del ejemplo, ya que provee mecanismos para una fácil incorporación de las mismas.
Otra ventaja del VISPRO C++, es que permite utilizar IGNATIUS, con solo invocar esta biblioteca en la linkeditción del código generado.

Otras alternativas analizadas fueron VisualBuilder del VisualAge C++ 3.0, y Dialog. Se encontró que estas herramientas son más complejas de utilizar por programadores no experimentados, los destinatarios de la herramienta.

1.6 Desarrollo de la IGNATIUS

En esta sección detallamos cuestiones básicas relacionadas con el desarrollo de la herramienta. Detalles de desarrollo e implementación (documentación, código fuente, diseño etc.), puede encontrarse en la segunda sección de la tesis (Informe Técnico).

Para comenzar, luego de haber analizando varios supervisores disponibles, identificamos tres niveles de servicios comunes.

1. *Servicios de Interfaz:* son aquellas rutinas que proveen la interfaz entre el Supervisor y el operador. Poseen típicamente displays gráficos, alarmas audibles, mensajes por pantalla, etc. La implementación de estas rutinas está íntimamente relacionada con la plataforma de desarrollo elegida.

2. *Servicios Particulares:* son aquellas rutinas que implementan los requerimientos particulares de un SCADA específico, como por ejemplo: rutinas de atención de alarmas, manejadores de eventos, rutinas de atención de comandos de usuario, etc. La implementación de este servicio depende del entorno que se desea modelar.

3. *Servicios Básicos:* son aquellas rutinas que implementan los servicios básicos de supervisión, comunes a todos los supervisores. Por ejemplo: representación interna de los puntos de la imagen, detección de condiciones de alarma, almacenamiento histórico, lectura y escritura en los ports de comunicaciones, ejecución de comandos del operador, planificación de tareas, modelado de los procesos del mundo exterior, etc.

La Figura 17 describe gráficamente la arquitectura de una aplicación de Supervisión y Control (SCADA), también denominada Supervisor dentro de este documento, que usa IGNATIUS:

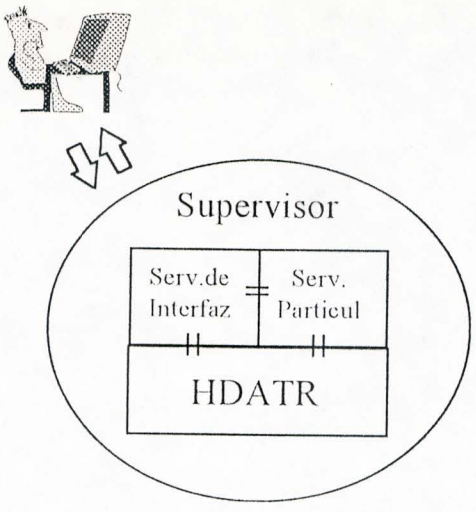


Figura 17
Arquitectura de un Supervisor

El trabajo propuesto se basa en diseñar y construir el 3er nivel de servicio, de aquí en mas denominado IGNATIUS.

1.6.1 Qué es IGNATIUS ?

IGNATIUS es una herramienta orientada a facilitar y minimizar la tarea de programación de aplicaciones SCADA a través de elementos y servicios comunes a toda aplicación de Supervisión y Control. Usado la clasificación de [Kap95], podemos decir que IGNATIUS es una Interfaz con un Lenguaje de Programación. Como vimos este tipo de SCADA tiene la ventaja de ser muy flexible ya que son librerías que se adaptan a cualquier proceso que se desee controlar. No está orientado a usuarios finales, sino a programadores que deben desarrollar la aplicación específica, pero como se verá mas adelante, usando IGNATIUS esto puede hacerse con mínimo esfuerzo. Está implementada como una librería de clases escrita en C++.

Fue desarrollado con el concepto de Cliente/Servidor en el cual IGNATIUS es un servidor y los niveles 1 y 2 son los clientes que varían para cada Supervisor en particular. Por lo tanto, podríamos pensar a un único servidor dando servicios a distintos clientes que pueden variar de acuerdo a distintos requerimientos funcionales o de hardware.

IGNATIUS provee a través de sus clases básicamente tres servicios:

- *Tablas de Representación.* En ellas se almacenan los valores de representación de los distintos objetos del mundo real como ser:
  - Imagen
  - Alarmas
  - Procesos
  - Ports
  - Tareas
  - Mímicos
  - Comandos
  - Colas de Entrada y Salida
  - Histórico

Las estructuras y los datos se encuentran encapsulados dentro de cada clase. La implementación de estas tablas depende de la naturaleza del objeto a representar (Ver Elección de las Estructuras Internas de Almacenamiento de los Datos en *Informe Técnico*).

- *Métodos de Acceso a la Información de las Clases.* La forma de acceder a los datos que se encuentran almacenados en las tablas de representación es a través de los métodos de acceso. Los métodos de acceso son los métodos de las clases - funciones - que permiten al usuario grabar, borrar, modificar y leer la información de las tablas de representación. Por otra parte, es la única manera de acceder a los datos ya que los mismos están encapsulados dentro de las clases

- *Motor de Ejecución.* El motor de ejecución es el conjunto de métodos que ponen en funcionamiento a la aplicación. Estas rutinas ejecutan concurrentemente dentro de distintos threads coordinadas por una rutina de planificación. Están sincronizadas usando semáforos de evento. Los siguientes métodos componen el Motor de Ejecución:
  - Planificador de Tareas
  - Analizador de Alarmas
  - Actualizador de Histórico
  - Administrador de Mímicos
  - Intérprete de Comandos
  - Manejador de Ports

1.6.2 Descripción Funcional de las Clases

Las clases que componen IGNATIUS son las siguientes:

- *TABLEAUX*. Esta clase representa los valores de los puntos de la imagen del mundo exterior y las funciones intrínsecas a los mismos. Todos los valores de los puntos monitoreados, que representan el estado de la planta están almacenados en esta clase. Los puntos pueden ser enteros, reales o digitales. Por cuestiones de implementación, se dividió a la imagen en 3 segmentos virtuales que representan los valores enteros, analógicos y digitales. Por este motivo se definió a la clase TABLEAUX como una clase abstracta que es heredada por las clases TABLEAUX_INT, TABLEAUX_DOUBLE y TABLEAUX_DIG, que representan a cada uno de los tipo que puede tener un punto de la imagen. Este aspecto es transparente en tiempo de ejecución ya que los puntos de la imagen son tratados independientemente del tipo de datos de los mismos.

- *ALARMA.* Esta clase representa las condiciones de alarma que deberán ser evaluadas por el motor de ejecución y las funciones intrínsecas a las mismas. Relaciona condiciones de alarma con los puntos de la imagen asociados a las mismas. Este esquema permite definir libremente relaciones de tipo muchos-a-muchos entre las condiciones de alarma y los puntos de la imagen representados por el sistema. Las rutinas de atención de alarmas de los servicios particulares pueden analizar la información de los estados de alarma detectados por el motor de ejecución para darle el tratamiento adecuado y específico del problema.

- *COLA.* Esta clase es usada para representar los valores que van desde IGNATIUS hacia el mundo exterior como así también los valores que ingresan a IGNATIUS desde el mundo exterior. Sirve para comunicar los PLC o conversores AD/DA con el supervisor. Implementa una lista encadenada de elementos con política FIFO (primero en entrar es primero en salir)

- *HISTÓRICO.* Esta clase es la encargada de almacenar, leyendo los mensajes almacenados en la cola de entrada, los registros históricos de actualización de los valores de la imagen en función de parámetros especificados por el usuario. Esta información puede ser utilizada para analizar y detectar cambios y problemas producidos en el entorno, ya que permite desplegar información de

los distintos valores que fue tomando un determinado punto de la imagen a través del tiempo. La frecuencia con la que se almacena en este archivo el valor de un punto depende de un porcentaje de variación establecido de antemano por el usuario para ahorrar espacio de almacenamiento.

- *TAREA.* Esta clase es la encargada de planificar la ejecución de los distintos componentes del motor de ejecución. Su uso puede extenderse para planificar rutinas escritas por el usuario, como por ejemplo una rutina de refresco de datos en pantalla. Almacena toda la información de las tareas del motor de ejecución, excepto la del Planificador de Tareas que está encapsulado en esta clase. El Planificador de Tareas es el encargado de usar esta información para sincronizar la ejecución de cada una de las tareas.

- *PROCESO.* Esta clase es usada para representar los distintos procesos del mundo exterior, como por ejemplo: proceso de secado de materia prima textil, proceso de llenado y vaciado de fluidos, etc. Esta información puede ser usada por los Servicios de Interfaz para las funciones de monitoreo por pantalla.

- *MSG.* Esta clase es la encargada de encapsular los mensajes que envía la clase *MIMICO* al supervisor en un pipe. Estos mensajes contienen información de la representación de los puntos de la imagen en la pantalla. Esta información es procesada por los Servicios de Interfaz del supervisor y desplegada por pantalla con el formato de representación especificado.

- *PIPE.* Esta clase es la encargada de implementar el canal de comunicaciones entre la clase *MIMICO* y el supervisor para el envío de mensajes (objetos de la clase *MSG*). La clase *MIMICO* escribe en el PIPE los mensajes mientras que los Servicios de Interfaz del supervisor los leen para usar la información de los mismos para el monitoreo por pantalla de los procesos del mundo exterior.

- *MÍMICO.* Esta clase es usada para relacionar los puntos de la imagen con los distintos procesos del mundo exterior y sus valores de representación: posición en la pantalla, color de letra, color de fondo, etc. La rutina del Administrador de Mímicos se encarga de escribir en el pipe los valores del proceso que se está monitoreando, con una frecuencia definida para ese proceso. Esta clase está compuesta, además, por dos clases de implantación: PIPE y MSG. Estas últimas sirven para establecer la comunicación entre el Administrador de Mímicos y el supervisor.

- *PORT.* .Esta clase sirve para manejar los ports de comunicaciones a través de los cuales se intercambian datos entre IGNATIUS y el mundo exterior. También relaciona a los distintos dispositivos físicos con los puntos de la imagen asociados, permitiendo un esquema de relación de tipo muchos-a-muchos. Fija los parámetros para establecer la comunicación con los ports (velocidad de transmisión, control de paridad, bits de datos, etc.). Lee la cola de salida del supervisor y escribe los mensajes en los ports asociados. Lee cada uno de los ports definidos a la espera de un mensaje, y cuando lo recibe lo escribe en la cola de entrada al supervisor.

- *COMANDO.* Esta clase provee al usuario un mecanismo para definir y administrar los comandos ingresados por el operador. Además posee un mecanismo de validación que permite determinar si un comando ingresado por el operador es válido y si los parámetros especificados son correctos. La ejecución de las rutinas de atención de los comandos puede ser llevada a cabo de dos formas: planificada o inmediata. En el primer caso, cada comando ingresado por el operador es agregado a una cola de comandos y atendido luego de haber atendido los comandos ingresados anteriormente. En el segundo caso, el comando ingresado es atendido primero, independientemente de los comandos que se encuentren encolados.

1.6.3 Ejemplo de un Supervisor que usa IGNATIUS

Para realizar una prueba total de la herramienta se construyó un supervisor para un horno de secado El horno consta de 3 áreas: precalentado, secado y enfriado. El material a ser secado es ubicado en una cinta transportadora, la cual los transporta a través del horno. La velocidad normal de la cinta es entre 250 y 300 cm/min. Si la velocidad cae por debajo de estos valores se emite una señal de alarma. El horno es calentado por tres mecheros de gas, distribuidos a intervalos regulares. La temperatura en cada una de las áreas calentadas por los mecheros, es monitoreada y controlada por el operador a través de una consola. Si la temperatura para un área varía mas del 5% de su punto de referencia se emite una señal de alarma. Cada una de las áreas es controlada usando un algoritmo de control PID. El requerimiento es reemplazar el sistema de control cableado por un sistema basado en una computadora. En la Figura 18, vemos un esquema general del Horno de Secado.

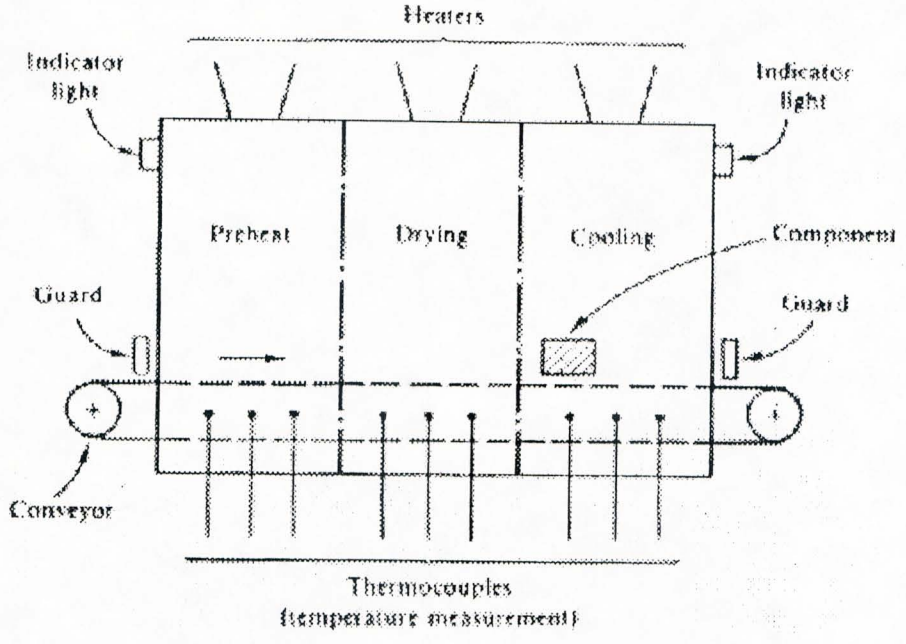


Figura 18
Esquema General del Horno de Secado

El operador puede emitir los siguientes comandos:

- START: enciende los mecheros y se enciende la cinta
- RESET: ocasiona la baja de todo el sistema.
- RE-START: provoca que la cinta pare
- STOP: la respuesta a este comando será apagar todos los mecheros y detener la cinta.

Para la implementación de este ejemplo usando IGNATIUS se definieron cuatro alarmas. Tres de ellas encargadas de monitorear el estado de las distintas áreas del horno y la cuarta dedicada al monitoreo de la velocidad de la cinta. Se definió una interfaz con el operador flexible con menúes desplegables que le permite configurar la herramienta para un uso específico. Permite al operador realizar las siguientes acciones :

- Definir los puntos de la imagen y la actualización de los mismos
- Definir procesos y actualizarlos, asociar procesos con bitmaps
- Definir de mímicos y su asociación con puntos de la imagen

- Actualizar alarmas, definir nuevas, o modificar alarmas ya existentes. Permite la asociación de una alarma con un proceso definido por el usuario, para su atención
- Permite armar y mantener relaciones entre las alarmas y puntos de la imagen que se desea controlar
- Definir ports
- Asociar ports con puntos de la imagen, esto permite saber al proceso residente de donde obtener la información para la actualización de un port determinado
- Vizualizar el archivo de datos históricos, lo que permite al operador conocer como estuvo funcionando la planta.
- Arrancar el motor de ejecución, con esto da comienzo a la aplicación de supervisión.
- Ingresar comandos a través de una línea de comandos. Estos comandos deberán estar relacionados con procesos definidos por el operador.

La base de dicho supervisor se construyó usando IGNATIUS. A continuación se describen como están compuestos los distintos servicios.

1.6.3.1 Servicios de Interfaz (Nivel 1)

Para la codificación de los *servicios de interfaz* se utilizó una herramienta de programación visual que agilizó notablemente la construcción de las ventanas y todos los objetos gráficos.

Esta interfaz está compuesta por los siguiente objetos:

- *Main.* Este objeto es el que provee la ventana principal del supervisor. Entre principales funciones que realiza se encuentra la creación de objetos de IGNATIUS (imagen, alarma, histórico, etc.), inicialización de variables, definición de las rutinas de atención de comandos, creación de la ventana principal del supervisor y de todos los objetos que la componen (botones, barra de acción, campo de ingreso de datos, etc.), creación del pipe de comunicaciones, creación de un prompt para el ingreso de comandos, planificación de las Tareas, lanzamiento y detención del motor de ejecución e invocación de las restantes ventanas de la aplicación según selección del usuario.

- *VpAlarma.* Este objeto es el que provee al operador un front-end para la actualización y monitoreo de las condiciones de alarma. Las funciones llevadas a cabo por este objeto comprenden: inicialización de variables, definición de las rutinas de atención de alarma, creación de los objetos de la ventana de actualización de alarmas, visualización de las alarmas existentes, actualización de alarmas.

- *VpAlarmaImagen.* Este objeto es el que provee al operador un front-end para la actualización y monitoreo de las relaciones entre las condiciones de alarma y los puntos de la imagen. Sus funciones incluyen inicialización de variables, creación de los objetos de la ventana de relaciones, visualización de las alarmas existentes, visualización de los puntos de la imagen existentes, visualización de las relaciones entre las condiciones de alarma y los puntos de la imagen y actualización de las relaciones.

- *VpHistorico. Es* el que provee al operador un front-end para monitorear e imprimir los registros almacenados en el archivo histórico. Sus funciones incluyen inicialización de variables, creación de los objetos de la ventana de registros del archivo histórico, visualización de los registros del archivo histórico e impresión de los registros del archivo histórico.

- *VpImagen.* Es el que provee al operador un front-end para la actualización y monitoreo de los valores de los puntos de la imagen. Las funciones llevadas a cabo por este objeto son: inicialización de variables, creación de los objetos de la ventana de actualización de imagen, visualización de los puntos de la imagen existentes, actualización de los puntos de la imagen.

- *VpMimico*. Este objeto es el que provee al operador un front-end para la actualización y monitoreo de la representación de los procesos y los valores de los puntos de la imagen asociados a los mismos (mímicos). Las funciones que incluye son : inicialización de variables, creación de los objetos de la ventana de mímicos, visualización de los puntos de la imagen existentes, visualización de los procesos existentes, visualización de los mímicos existentes y actualización de los mímicos.

- *VpPort*. Es el que provee al operador un front-end para la declaración y monitoreo de los ports conectados a los dispositivos. Sus funciones son inicialización de variables, creación de los objetos de la ventana de ports, visualización de los ports declarados y actualización de las declaraciones de ports.

- *VpPortImagen*. Es el que provee al operador un front-end para la actualización y monitoreo de las relaciones entre los ports conectados a los dispositivos y los puntos de la imagen. Entre sus principales funciones se incluyen inicialización de variables, creación de los objetos de la ventana de ports, visualización de los ports declarados y actualización de las declaraciones de port.

- *VpProceso*. Es el que provee al operador un front-end para la actualización y monitoreo de los procesos del mundo exterior. Entre sus principales funciones se encuentran inicialización de variables, creación de los objetos de la ventana de actualización de procesos, visualización de los procesos existentes, actualización de los procesos, monitoreo en la ventana *Main* del proceso seleccionado para monitorear, selección visual de la imagen asociada al proceso.

1.6.3.2 Servicios Particulares (Nivel 2)

Los *servicios particulares* fueron implementados como programas C++. Los tres programas que componen estos servicios son los siguientes:

- *USERHDR.CPP*. Este programa es el manejador de mensajes del usuario. Su función principal es capturar los mensajes enviados a la ventana Main por los objetos *MIMICO*, *vpProceso* y por las rutinas de atención de comandos.

- *PROC_USU.CPP*. Este programa contiene las rutinas de atención de alarmas codificadas por el usuario. Estas rutinas son ejecutadas en un thread independiente por el Analizador de Alarmas.

- *USERCMD.CPP*. Este programa contiene las rutinas de atención de comandos codificadas por el usuario.

1.7 Resultados Obtenidos

A los efectos de demostrar el uso de la herramienta se construyeron dos sistemas SCADA que describiremos a continuación.

El primero, que de aquí en mas llamaremos "ejemplo 1", demuestra el uso de la herramienta empleada para la construcción de un sistema SCADA configurable por el operador. Este SCADA le permite al operador la configuración dinámica de los recursos: definir los puntos de la imagen, definir las condiciones de alarma, definir los procesos que se quieren monitorear, los dispositivos especiales que usan los procesos, definir la representación por pantalla, actualizar los valores de los puntos de la imagen, actualizar las condiciones de alarma, etc. La flexibilidad dada por la configuración dinámica, permite aplicar el SCADA a cualquier uso particular.

El segundo, que de aquí en mas llamaremos "ejemplo 2", demuestra el uso de la herramienta empleada para la construcción de un sistema SCADA de aplicación específica. En este caso los recursos están definidos estáticamente no permitiendo al operador la configuración de los mismos.

En ambos ejemplos, la aplicación configurada corresponde al proceso de un horno de secado. La especificación de este proceso trata de automatizar la operación de un horno dividido en tres sectores: precalentado, secado y enfriado. Los componentes a secar son colocados sobre una cinta transportadora que los hace pasar lentamente a través de los tres sectores del horno. Las temperaturas de los distintos sectores del horno son monitoreadas constantemente para mantenerlas dentro de los rangos preestablecidos.

Para ambos casos, la lectura de la información de los dispositivos (sensores de temperatura), se hizo a través de un emulador implementado a tal efecto, por no disponer del hardware apropiado para este fin. Este emulador de dispositivos fue implementado como un proceso independiente que escribe en el puerto de comunicaciones del equipo. Cada cierto intervalo de tiempo, se escriben mensajes como los que espera recibir el *Manejador de Ports* de la herramienta. Este es el encargado de leer estos puertos, armar el mensaje y colocarlo en la cola de entrada al supervisor. Decidimos construir otro emulador que nos permitiera probar el funcionamiento de la herramienta (con excepción de la clase PORT) con el hardware disponible. Este emulador de dispositivos fue implementado como una clase cuya tarea principal es escribir en la cola de entrada al supervisor un patrón de valores especificados cada cierto intervalo de tiempo. Esta tarea, si bien no es parte de IGNATIUS, está sincronizada por el *Planificador de Tareas*; esto permitió además, demostrar la utilización del planificador de alto nivel de IGNATIUS para sincronizar tareas del usuario. Cada uno de los sensores fue emulado a través de un objeto de esta clase y el patrón de valores y frecuencia para cada uno de ellos fue establecido durante la construcción del supervisor.

El ejemplo 1, permitió medir la utilidad de la herramienta empleada para la construcción de un sistema SCADA complejo, como así también mostrar el esfuerzo en construir un SCADA con todas las facilidades permitidas por la herramienta. La tarea de configuración dinámica resultó sencilla y rápida. El ejemplo 2, permitió medir la utilidad de la herramienta empleada para la construcción de un sistema SCADA de características simples y también determinar el esfuerzo mínimo que requiere construir un SCADA usando la herramienta.

Sometimos a los SCADAs de ejemplo a pruebas en las cuales la CPU era compartida por otras aplicaciones, cambiando su modo de ejecución a background.

A pesar de que la CPU no estaba dedicada a esta tarea, el tiempo de respuesta del supervisor resultó dentro de los parámetros preestablecidos gracias a los algoritmos de planificación del Sistema Operativo y al *Planificador de Tareas* de la herramienta.

La política del sistema operativo de multitareas con desalojo, permitió asegurar que ningún proceso iba a hacer uso excesivo de la CPU retrasando la ejecución de los procesos bloqueados a la espera de CPU. Las tareas del Motor de Ejecución de los SCADAs fueron planificadas con distintos intervalos de tiempo dentro del *Planificador de Tareas*. Se comenzó planificando a las tareas despertándolas con intervalos superiores al segundo y observando que los tiempos se respetaban considerando una carga media del sistema. Finalmente, se observó el intervalo mínimo para que, con una carga media, el sistema responda dentro de los tiempos esperados:

- *Analizador de Alarma* 0.15 seg
- *Actualizador de Histórico* 0.45 seg.
- *Administrador de Mímicos* 0.35 seg.
- *Intérprete de Comandos* 0.65 seg.
- *Manejador de Ports* 0.55 seg.

Esto permitió realizar otras tareas durante la ejecución del supervisor como por ejemplo: analizar los datos históricos, graficar nuevos procesos en formato representable por el supervisor (bitmaps en nuestro ejemplo), lanzar un shell de comandos, agregar condiciones de alarma, procesos, mímicos, etc., permitiendo compartir la CPU con otras tareas sin la necesidad de detener el funcionamiento del supervisor.

Los tiempos usados para diseñar, codificar y probar los SCADAs de ejemplo desarrollados fueron mínimos. En el caso del supervisor del ejemplo 1, el esfuerzo fue de 120 horas hombre, mientras que para el segundo ejemplo se usaron tan sólo 24 horas hombre. Estos valores reflejan claramente la utilidad de la herramienta y el ahorro en los tiempos totales de desarrollo de un sistema supervisor.

Para acelerar los tiempos de ejecución se podría pensar en embeber el sistema operativo junto con el supervisor desarrollado en una EPROM. Para esto hubiera sido conveniente haber desarrollado sobre QNX, ya que es un sistema operativo totalmente modular y hubiera permitido embeber únicamente los módulos de utilidad al supervisor y no la totalidad del sistema operativo. No pasa lo mismo con OS/2 que es un sistema operativo monolítico. A pesar de esto la disminución de costos del hardware hace factible pensar en esta posibilidad.

Se hizo difícil el debugging de algunos errores dentro del código debido a la cantidad de tareas concurrentes dentro del motor de ejecución, aunque el diseño orientado a objetos de la herramienta y la potencia del debugger del OS/2, facilitaron esta tarea.

Con respecto a la configuración del hardware usada para implementar la herramienta, inicialmente usamos un procesador 486 con 4 Mb de memoria RAM y 120 Mb de espacio en disco. Esto no era suficiente para usar el compilador VisualAge C++ para OS/2, debido a que es recomendable disponer de un mínimo de 16 Mb de memoria RAM y 300 Mb de espacio en disco. Por este motivo, y hasta tanto conseguimos ampliar el hardware de nuestro equipo, decidimos comenzar la codificación de la herramienta usando el Turbo C++ para DOS que podía ejecutar con los recursos de hardware que disponíamos en ese momento. Nos concentramos en codificar las *Tablas de Representación* y los *Métodos de Acceso a las Clases,* postergando para una etapa posterior la codificación del *Motor de Ejecución*, ya que esta última es la parte de la herramienta que se encuentra más ligada al sistema operativo. Poco antes de finalizar la codificación de estas dos partes, conseguimos ampliar el hardware e instalar OS/2 Warp 3.0 y VisualAge C++ 3.0 para OS/2. La tarea de migración del código a esta nueva plataforma resultó prácticamente nula y ahora disponíamos de nuevas facilidades: programación multitarea, un debugger muy potente, gran cantidad de documentación embebida, editor inteligente, etc. lo que aumentó nuestra productividad notablemente y nos permitió finalizar la codificación de la herramienta.

De acuerdo a la descomposición en niveles de los sistemas de supervisión, se puede observar que la implantación de los Servicios de Interfaz y Servicios Particulares varía para cada supervisor. Sin embargo, la implantación de los Servicios Básicos podría ser única para cualquier supervisor ya que es independiente de la plataforma de software, hardware o del problema que se desea modelar.

Debido a la existencia en el mercado de múltiples herramientas de programación visual, orientadas a acortar el ciclo de desarrollo de aplicaciones, acelerando y simplificando la construcción de las interfaces gráficas, el desarrollo de los Servicios de Interfaz se ve notablemente reducido en tiempo y complejidad.

El costo de desarrollo de los Servicios Particulares es proporcional a la complejidad del entorno que se desea modelar. Como hemos podido comprobar, para entornos de complejidad media, el costo es mínimo.

Con el enfoque propuesto y debido a que los Servicios Básicos encapsulan la mayor parte de la funcionalidad de un supervisor, si se desea encarar el desarrollo de cualquier sistema supervisor, el

trabajo se limita a diseñar y codificar los Servicios de Interfaz y los Servicios Particulares. De esta forma, la mayor parte del desarrollo ya está resuelta.

1.8 Conclusiones y Trabajos Futuros

Actualmente, existe un gran número de computadoras en la industria controlando procesos y ejecutando aplicaciones de tiempo real. Desde los comienzos del uso de las computadoras para control de procesos del mundo real, las técnicas de control retroalimentado de la planta han cambiado muy poco; es en el rol de supervisión de procesos donde han ocurrido la mayoría de los cambios.

Sin embargo, las aplicaciones de supervisión suelen ser poco flexibles y tienen un rango de utilización muy limitado. Existen pocos Sistemas Operativos y lenguajes de programación que provean facilidades relacionadas con la ejecución de tareas con restricciones duras de tiempo. Tampoco existen aún demasiados entornos de desarrollo que solucionen todos los problemas relacionados con restricciones en este tipo de sistemas, y, por ende, las aplicaciones siguen construyéndose "ad hoc" para cada uno de los problemas.

Este panorama nos motivó para la realización de nuestro trabajo, que se basa en el diseño e implantación de una herramienta que permite el desarrollo de un grupo importante de aplicaciones de Tiempo Real (las de Supervisión de Procesos), facilitando el trabajo de los diseñadores. También nos motivó el interés en investigar un área de las ciencias de la computación, en la cual se haya incursionado mínimamente en nuestro país.

Se realizó una descomposición interna de los sistemas supervisores en tres niveles de servicio, encapsulando el nivel de Servicios Básicos dentro de la herramienta.

Se realizó un estudio exhaustivo de diversas metodologías de diseño resultando MASCOT la más apropiada para el diseño de la herramienta. También se realizó una cuidadosa selección del Sistema Operativo y el lenguaje de programación apropiados para la construcción de la herramienta, resultando OS/2 y C++ los elegidos. El C++ nos permitió implementar a la herramienta como un conjunto de clases sin jerarquía entre sí, elegibles para la construcción de cualquier sistema supervisor.

Se implementaron, a modo de ejemplo, 2 SCADAs utilizando la herramienta desarrollada. Estos ejemplos permitieron demostrar la utilidad de la herramienta y el esfuerzo requerido para el desarrollo de sistemas supervisores de distinta complejidad. Se sometieron estos ejemplos a pruebas de performance, midiendo los tiempos de respuesta obtenidos para distintos períodos de intervalo de las tareas.

El uso de objetos facilitó el diseño de la herramienta ya que fue natural pensar en los distintos elementos que componen los Servicios Básicos como objetos independientes. También facilitó la detección de errores en el código debido al encapsulamiento de datos y funciones asociadas. A su vez provee características de reusabilidad y mantenibilidad de código propias del paradigma.

La elección de OS/2 Warp 3.0 como Sistema Operativo, proveyó un entorno de desarrollo probado y estable, ya que debido a su ejecución en modo protegido es difícil provocar la caída del mismo debido a errores involuntarios en la codificación de las aplicaciones. Su característica de multitarea aumentó nuestra productividad durante el desarrollo, debido a que nos permitió realizar varias tareas simultáneamente, sin la necesidad de finalizar y volver a iniciar un proceso cada vez que necesitábamos interrumpirlo; por ejemplo: nos permitió realizar debugging de la herramienta y simultáneamente corregir errores que íbamos detectando, sin la consecuente pérdida de tiempo que hubiera implicado finalizar la tarea de debugging para iniciarla nuevamente una vez corregido el error encontrado.

Un problema que encontramos en el uso del OS/2 es el siguiente: este sistema operativo se maneja a través de mensajes; cada vez que se produce un evento en el sistema se dispara un mensaje que es manejado por un único manejador de mensajes. Esto quiere decir que existe una única cola de mensajes para todas las aplicaciones que ejecutan en el sistema operativo. Si bien el OS/2 provee facilidades para manejar la cola de mensajes, como ser establecer su tamaño, etc., debido a que el manejador de la cola de mensajes es único para todo el sistema, se pueden provocar situaciones en las cuales existen muchos procesos enviando mensajes al manejador de mensajes, no dando tiempo a éste a remover los mensajes de la cola lo suficientemente rápido como para evitar que ésta se llene y se produzca la caída del sistema. Este inconveniente se puede evitar sincronizando al proceso que envía el mensaje y al que lo recibe, de manera que el que envía, lo haga recién después de haber recibido la confirmación que el receptor procesó el mensaje anterior.

Otro inconveniente que encontramos en el uso del OS/2, es que éste no permite a los programas de aplicación realizar operaciones directamente sobre los dispositivos físicos. Los device drivers actúan como interfaz entre los programas de aplicación y el hardware del equipo. Para acceder un dispositivo a través de una aplicación se debe usar el device driver correspondiente, lo que si bien aísla al programa de aplicación de las cuestiones físicas del hardware, lo hace dependiente del device driver que se haya cargado para manejar el dispositivo; por ejemplo, si se quiere escribir en la consola, el resultado de la operación de escritura puede variar, dependiendo si el device driver que maneja la consola es ANSI o no. Algunos periféricos están disponibles a las aplicaciones sólo si el device driver apropiado fue instalado previamente; por ejemplo, una aplicación no puede abrir el port serial a menos que el device driver de comunicaciones haya sido cargado. La única manera de acceder directamente a un dispositivo físico es escribiendo un device driver que lo maneje.

La instalación del VisualAge C++ 3.0 trajo problemas de compilación del código generado por el Vispro/C++ 1.0, para la construcción de los SCADAs de ejemplo, debido a que el compilador es más reciente que el generador de código. Este problema no estaba mencionado dentro de la documentación del generador de código. Luego de analizar el problema, pudimos detectar que se debía a que el Vispro/C++ 1.0 no estaba preparado para usar las librerías del nuevo compilador. Logramos solucionarlo detectando cuáles eran las librerías que generaban el problema, y reemplazando en el nuevo compilador estas librerías por las de la versión más vieja.

Inicialmente, no disponíamos de la configuración de hardware necesaria para codificar en la plataforma de software elegida, OS/2 Warp y VisualAge C++ 3.0. Hasta que conseguimos ampliar la configuración de hardware de nuestro equipo y para no perder el tiempo, decidimos comenzar la tarea de programación de la herramienta con un compilador C++ para DOS. Esto hizo que nos concentráramos en aspectos de portabilidad del código. Una vez solucionado el inconveniente, logramos recompilar en la nueva plataforma, lo codificado de la herramienta hasta ese momento, con un mínimo de esfuerzo.

La programación en C++ y el desarrollo en OS/2 fue una tarea novedosa para nosotros, debido a que nunca habíamos codificado en ese lenguaje y a las particularidades que posee el sistema operativo. Los conceptos de programación orientada a objetos, si bien no eran nuevos para nosotros, nunca habían sido llevados a la práctica. El manejo de mensajes del sistema operativo, los conceptos de threads y la interfaz gráfica también representaron nuevos conocimientos.

Existe poca bibliografía en el mundo que hable de SCADAs, y mucha menos aun que hable desde el punto de vista del diseño de los mismos. La poca bibliografía existente, encara mayormente el estudio de la teoría de control o de sistemas de tiempo real descuidando los aspectos de supervisión. Esta problemática trasladada a nuestro país, se vuelve aun más aguda ya que la poca bibliografía que mencionamos es muy difícil de acceder.

Considerando lo expuesto anteriormente, podemos decir que los objetivos que nos propusimos fueron alcanzados debido a que la descomposición propuesta de los sistemas de supervisión en tres niveles de

servicio y la utilización de la herramienta para cubrir los Servicios Básicos, permiten limitar cualquier desarrollo de un sistema supervisor a la construcción de los Servicios Particulares y los Servicios de Interfaz, acortando notablemente los tiempos de desarrollo y minimizando la complejidad. Por otro lado, la herramienta desarrollada permite aislar al usuario de la resolución de restricciones de tiempo impuestas por este tipo de sistemas, favoreciendo la concentración en cuestiones de diseño de mas alto nivel (datos a ser mostrados por pantalla, rutinas de atención de alarmas, comandos del operador, etc.). Provee al usuario un entorno de desarrollo completo que puede ser utilizado por programadores no experimentados, favoreciendo la seguridad, mantenibilidad y correctitud de las aplicaciones desarrolladas. Finalmente, este trabajo será utilizado en los trabajos prácticos de un curso semestral de Introducción a los Sistemas de Tiempo Real, dictado en nuestro Departamento, y su integración con otros trabajos de licenciatura que puedan utilizarlo como sistema de información base para aplicaciones de Tiempo Real (en particular, se tratará de integrarlo con diversos trabajos de investigación y desarrollo en el marco del proyecto EX-033, "Concurrencia y Sistemas Operativos").

Como una ampliación a este trabajo podrían encararse los siguientes desarrollos:

1. Probar el grado de portabilidad de la herramienta migrándola a otros sistemas operativos.

2. Diseñar e implementar un sistema supervisor usando una arquitectura cliente/servidor. Los Servicios Básicos deberán definirse dentro del servidor para que corran en una determinada plataforma; mientras que los Servicios de Interfaz y los Servicios Particulares se definirán como clientes en una plataforma distinta a la del servidor.

3. Diseñar e implementar a los Servicios Básicos como un único servidor de distintos clientes (Servicios de Interfaz y Servicios Particulares) que representen distintos supervisores y por lo tanto distintas representaciones del mundo real, cada una de ellas corriendo en distintas plataformas.

4. Incorporar interfaces con distintas PCs industriales y PLCs, para facilitar su uso.

5. Probar la herramienta en entornos de aplicación rudos.

6. Probar de embeberla en memoria ROM.

7. Desarrollar nuevas aplicaciones de Supervisión más complejas: planillas de cálculo, generador de estadísticas, etc.

8. Desarrollar rutinas de IA para planificación inteligente.

Referenicias y Biliografía

[Bas] BASSET, E. "Real Time PC Control". Industrial Computing, February 1995, pp 14-17.

[Ben93] BENNET, S. "Real Time Computer Control: an Introduction". Prentice Hall International. 2nd Edition. 1993.

[Deu88] DEUTSCH, M. "Focusing Real Time Systems analysis on User Operations". IEEE Software. September 1988, pp 39-50.

[Gom84] GOMAA, H. "A Software Design Method for Real Time System". Communication of the ACM, September 1984, pp 938-949.

[Gom86] GOMAA, H. "Software Development of Real Time Systems". Communication of the ACM, July 1986, pp 657-668.

[Jak75] JACKSON, M. A. "Principles of Program Design". Academic Press, New York, 1975.

[Kap95] KAPLAN, G.; HOUSE, R.; "Data Adquisition Software for engineers and scientists". IEEE Spectrum, May 1995, pp 23-39.

[Lap93] LAPLANTE, P. "Real Time Systems. Design and analysis. An Engineer's handbook". IEEE Press. 1993.

[Mas87] MASCOT. "The official handbook of MASCOT, versión 3.1". Computing Division, RSRE, Malvern 1987.

[Miy93] MIYAGI, P.; PEREIRA RIBEIRO BARRETO, M.; SILVA J. "Domotica: Controle e Automacao, Volumen II. VI EBAI 1993.

[Orf92] ORFALI, R.; HARKEY, D.; "Client/Server Programming with OS/2 2.0 Van Nostrand Reinhold. 2nd Edition. 1992.

[Sad95] SADRE, A. "21st Century PC Control", Industrial Computing, May 1995, pp 16-17

[Wai93] WAINER, G. "SSDT: Una Herramienta para Desarrollar Sistemas Supervisores en Tiempo Real". Encuentro Chileno de Computación. La Serena, Chile, Octubre 1993, pp 44-52.

[Wai94] WAINER, G. "Introducción al desarrollo de Sistemas en Tiempo Real ". FTP://zorzal.dc.uba.ar/pub/materias/str/libro, 1994 (publicación on line).

[War86] WARD, J.; MELLOR, P. "Structured Development for Real Time Systems" Yourdon Press 1986.

[Wat93] Waterbury, R. "Shootout in Raleigh". Industrial Computing, March 1993, pp 14-18.

2. INFORME TECNICO

2.1 Introducción

IGNATIUS es una librería de clases de objetos escrita en C++. Este informe está orientado a los usuarios de IGNATIUS, brindándoles una completa descripción de la herramienta y de la forma de usarla. Los primeros capítulos describen funcionalmente a IGNATIUS. Luego se detalla la especificación de cada una de las clases y finalmente se muestra el desarrollo de un Supervisor usando IGNATIUS.

2.1.1 QUÉ ES IGNATIUS?

IGNATIUS es una herramienta orientada a facilitar y minimizar la tarea de programación de aplicaciones SCADA a través de elementos y servicios comunes a toda aplicación de Supervisión y Control. Está implementada como una biblioteca de clases escrita en C++.

IGNATIUS provee a través de sus clases básicamente tres servicios:

- ***Tablas de Representación.*** En ellas se almacenan los valores de representación de los distintos objetos del mundo real como ser:
  - Imagen
  - Alarmas
  - Procesos
  - Ports
  - Tareas
  - Mímicos
  - Comandos
  - Colas de Entrada y Salida
  - Histórico

  Las estructuras y los datos se encuentran encapsulados dentro de cada clase. La implementación de estas tablas depende de la naturaleza del objeto a representar (Ver Elección de las Estructuras Internas de Almacenamiento de los Datos).

- ***Métodos de Acceso a la Información de las Clases.*** La forma de acceder a los datos que se encuentran almacenados en las tablas de representación es a través de los métodos de acceso. Los métodos de acceso son los métodos de las clases - funciones - que permiten al usuario grabar, borrar, modificar y leer la información de las tablas de representación. Por otra parte, es la única manera de acceder a los datos ya que los mismos están encapsulados dentro de las clases.

- ***Motor de Ejecución.*** El motor de ejecución es el conjunto de métodos que ponen en funcionamiento a la aplicación. Estas rutinas ejecutan concurrentemente dentro de distintos threads coordinadas por una rutina de planificación. Están sincronizadas usando semáforos de evento. Los siguientes métodos componen el Motor de Ejecución:
  - Planificador de Tareas
  - Analizador de Alarmas
  - Actualizador de Histórico
  - Administrador de Mímicos
  - Intérprete de Comandos
  - Manejador de Ports

2.1.2 QUIÉN DEBE USAR IGNATIUS ?

IGNATIUS puede ser usado por aquellos integrantes de un grupo de desarrollo de sistemas responsables de la codificación de una aplicación SCADA.

El nivel de conocimiento requerido para el uso de la herramienta es el siguiente:

- Lenguaje de programación C++
- Programación en OS/2

2.2 Descomposición de IGNATIUS en Niveles de Servicios

La siguiente figura describe gráficamente la arquitectura de una aplicación de Supervisión y Control (SCADA), también denominada Supervisor dentro de este documento, que usa IGNATIUS:

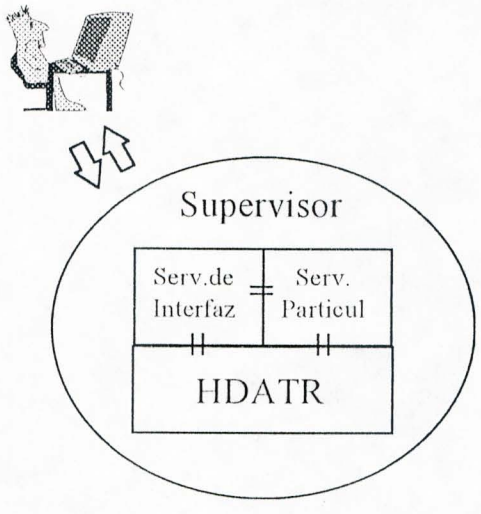


Figura 1. Arquitectura

El SCADA está compuesto por tres niveles:

1. *Servicios de Interfaz:* son aquellas rutinas que proveen la interfaz entre el SCADA y el operador. Poseen típicamente displays gráficos, alarmas audibles, mensajes por pantalla, etc.

Se comunican con IGNATIUS a través de los métodos de las clases, pipes, mensajes, etc.

Estos servicios no son provistos por IGNATIUSpor los siguientes motivos:

- La interfaz con el operador debe ser diseñada para cada Supervisor en particular de acuerdo a los requerimientos específicos del caso.

- Están íntimamente relacionados con la plataforma de HW y SW elegida para ejecutar el Supervisor (ej.: los bitmaps de OS/2 no poseen el mismo formato que los de Windows, lo mismo sucede con los

iconos, las sentencias para dibujar gráficos en una pantalla varían de acuerdo al SO en el cual se está programando, etc.).

- Existe una gran variedad de herramientas de cuarta generación o de programación visual que permiten generar esta interfaz de manera rápida y sencilla.

2. *Servicios Particulares:* son aquellas rutinas que implementan los requerimientos particulares del Supervisor, como por ejemplo: rutinas de atención de alarmas, manejadores de eventos, rutinas atención de comandos de usuario, etc.

Estas rutinas no son comunes a todos los Supervisores, por lo tanto deben ser diseñadas para cada Supervisor en particular de acuerdo a los requerimientos específicos del caso. Por este motivo son provistas por IGNATIUS

3. *Servicios Básicos:* son aquellas rutinas que implementan los servicios básicos de supervisión, comunes a todos los supervisores. Por ejemplo: representación interna de los puntos de la imagen, detección de condiciones de alarma, almacenamiento histórico, lectura y escritura en los ports de comunicaciones, ejecución de comandos del operador, planificación de tareas, modelado de los procesos del mundo exterior,etc. Estos servicios son los provistos por la herramienta IGNATIUS (ver Descripción funcional de las Clases).

IGNATIUS fue desarrollado con el concepto de Cliente/Servidor en el cual IGNATIUS es un servidor y los niveles 1 y 2 son los clientes que varían para cada Supervisor en particular. Por lo tanto, podríamos pensar a un único servidor dando servicios a distintos clientes que pueden variar de acuerdo a distintos requerimientos funcionales o de HW. Por ejemplo, para un Supervisor que ejecuta en modo gráfico de OS/2 (Presentation Manager) o en modo caracter dentro de una ventana de OS/2, lo único que cambia es el nivel 1.

2.3 Descripción Funcional de IGNATIUS

En este capítulo brindaremos una detalle de toda la funcionalidad brindada por IGNATIUS

2.3.1 Clases

La herramienta está compuesta por un conjunto de clases independientes que pueden ser utilizadas en la construcción de cualquier sistema supervisor. A continuación se detallan cada una de las clases de IGNATIUS.

2.3.1.1 TABLEAUX

Esta clase representa los valores de los puntos de la imagen del mundo exterior y las funciones intrínsecas a los mismos. Todos los valores de los puntos monitoreados están almacenados en esta clase.

Por cuestiones de implementación, se dividió a la imagen en 3 segmentos virtuales que representan los valores enteros, analógicos y digitales. Por este motivo se definió a la clase TABLEAUX como una clase abstracta que es heredada por las clases TABLEAUX_INT, TABLEAUX_DOUBLE y TABLEAUX_DIG. Este aspecto es transparente en tiempo de ejecución ya que los puntos de la imagen son tratados independientemente del tipo de datos de los mismos.

La clase contiene la siguiente información de cada uno de los puntos de la imagen representados:

- Valor
- Valor de Set-Point
- Estado
- Cantidad de referencias
- Porcentaje de modificación para actualización en el Histórico
- Fecha y hora de ultima actualización

Las funciones asociadas a la imagen son:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar un punto
- Borrar un punto
- Leer un punto
- Modificar el valor de un punto
- Sumar uno a la cantidad de referencias hechas a un punto
- Modificar el estado de un punto
- Obtener la cantidad de puntos
- Obtener el tipo de segmento
- Reconocer un punto en alarma

2.3.1.2 ALARMA

Esta clase representa las condiciones de alarma que deberán ser evaluadas por el motor de ejecución para detectar estados de alarma y las funciones intrínsecas a las mismas. Relaciona condiciones de alarma con los puntos de la imagen asociados a las mismas. Este esquema permite definir libremente relaciones de tipo muchos-a-muchos entre las condiciones de alarma y los puntos de la imagen representados por el sistema.

Las rutinas de atención de alarmas de los servicios particulares pueden analizar la información de los estados de alarma detectados por el motor de ejecución para darle el tratamiento adecuado y específico del problema.

Los datos almacenados por la clase para cada una de las condiciones de alarma son:

- Nombre
- Prioridad
- Dirección de la rutina de atención de alarma
- Valor mínimo
- Valor máximo
- Criterio de evaluación
- Cantidad de veces que se evaluó
- Frecuencia de evaluación
- Tiempo esperado desde la ultima evaluación
- Tasa máxima de cambios
- Tasa mínima de cambios
- Fecha y hora de ultima actualización

Los datos almacenados para cada una de las relaciones entre las condiciones e alarma y los puntos de la imagen son:

- Identificación de la alarma
- Índice del punto de la imagen relacionado
- Segmento de la imagen donde se encuentra el punto relacionado

Las funciones asociadas a las condiciones de alarma y sus relaciones son:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar una condición de alarma
- Modificar una condición de alarma
- Leer una condición de alarma
- Sumar uno a la cantidad de veces que se evaluó una condición de alarma
- Calcular los valores mínimo y máximo en función de un valor de set-point y de un porcentaje de tolerancia
- Buscar cuál es la condición de alarma relacionada a un punto de la imagen
- Buscar cuál es el punto de la imagen relacionado a una condición de alarma
- Relacionar una condición de alarma con un punto de la imagen
- Borrar una condición de alarma
- Borrar una relación
- Leer una relación
- Analizar condiciones de alarma
- Lanzar el Analizador de Alarmas
- Detener el Analizador de Alarmas

2.3.1.3 COLA

Esta clase es usada para representar los mensajes que van desde IGNATIUS hacia el mundo exterior como así también los mensajes que ingresan a IGNATIUS desde el mundo exterior. Sirve para comunicar los PLC o conversores AD/DA con el supervisor. Implementa una lista encadenada de elementos con política FIFO (primero en entrar es primero en salir)

Los datos almacenados en esta clase para cada uno de los mensajes son los siguientes:

- Segmento virtual de la imagen
- Índice del punto
- Valor
- Estado
- Fecha y hora de última actualización
- Puntero al próximo elemento

Las funciones asociadas a los mismos:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar un mensaje a la cola
- Sacar el primer mensaje de la cola

2.3.1.4 HISTÓRICO

Esta clase es la encargada de almacenar, leyendo los mensajes almacenados en la cola de entrada, los registros históricos de actualización de los valores de la imagen en función de parámetros especificados por el usuario.

Esta información puede ser utilizada para analizar y detectar cambios y problemas producidos en el entorno, ya que permite desplegar información de los distintos valores que fue tomando un determinado punto de la imagen a través del tiempo. La frecuencia con la que se almacena en este archivo el valor de un punto depende de un porcentaje de variación establecido de antemano por el usuario para ahorrar espacio de almacenamiento.

Los datos que se graban en el archivo histórico son los siguientes:

- Segmento virtual de la imagen
- Índice del punto
- Valor
- Estado
- Fecha y hora de última actualización

Las funciones asociadas a los mismos:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar un registro en el archivo histórico
- Comenzar la lectura del histórico
- Leer un registro del histórico
- Finalizar la lectura del histórico
- Listar el histórico
- Calcular la tendencia de un punto
- Actualizar histórico
- Lanzar el Actualizador de Histórico
- Detener el Actualizador de Histórico

2.3.1.5 TAREA

Esta clase es la encargada de planificar la ejecución de los distintos componentes del motor de ejecución. Su uso puede extenderse para planificar rutinas escritas por el usuario, como por ejemplo una rutina de refresco de datos en pantalla.

Almacena toda la información de las tareas del motor de ejecución, excepto la del Planificador de Tareas que está encapsulado en esta clase. El Planificador de Tareas es el encargado de usar esta información para sincronizar la ejecución de cada una de las tareas.

Los datos que se almacenan para cada una de las tareas son los siguientes:

- Nombre
- Índice de proceso
- Frecuencia de ejecución
- Tiempo esperado desde que ejecutó por última vez
- Manejador del semáforo de eventos de la tarea
- Fecha y hora de última actualización de la información

Las funciones asociadas a las tareas son:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar una tarea

- Modificar una tarea
- Leer una tarea
- Borrar una tarea
- Planificar las tareas
- Lanzar el Planificador de Tareas
- Detener el Planificador de Tareas

2.3.1.6 PROCESO

Esta clase es usada para representar los distintos procesos del mundo exterior, como por ejemplo: proceso de secado de materia prima textil, proceso de llenado y vaciado de fluidos, etc. Esta información puede ser usada por los Servicios de Interfaz para las funciones de monitoreo por pantalla.

Los datos que se almacenan para cada proceso son los siguientes:

- Nombre
- Identificación del proceso
- Identificación del objeto gráfico que lo representa
- Frecuencia de monitoreo
- Tiempo de espera se de la última vez que se monitoreó
- Descripción
- Fecha y hora de última actualización de la información

Las funciones asociadas son:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar un proceso
- Modificar un proceso
- Leer un proceso
- Borrar un proceso
- Modificar la frecuencia de monitoreo
- Validar si existe un proceso con una identificación dada

2.3.1.7 MSG

Esta clase es la encargada de encapsular los mensajes que envía la clase *MIMICO* al supervisor a través de un pipe. Estos mensajes contienen información de la representación de los puntos de la imagen en la pantalla. Esta información es procesada por los Servicios de Interfaz del supervisor y desplegada por pantalla con el formato de representación especificado.

Los datos encapsulados por esta clase son los siguientes:

- Longitud del mensaje
- Valor a representar
- Eje X de representación
- Eje Y de representación
- Color de letra
- Color de fondo

2.3.1.8 PIPE

Esta clase es la encargada de implementar el canal de comunicaciones entre la clase *MIMICO* y el supervisor para el envío de mensajes (objetos de la clase *MSG*). La clase *MIMICO* escribe en el PIPE los mensajes mientras que los Servicios de Interfaz del supervisor los leen para usar la información de los mismos para el monitoreo por pantalla de los procesos del mundo exterior.

Las funciones implementadas por esta clase son las siguientes:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Crear un pipe físico
- Abrir el pipe
- Leer el pipe
- Escribir en el pipe
- Conectar el pipe
- Desconectar el pipe
- Cerrar el pipe
- Enviar un mensaje a otro proceso

2.3.1.9 MÍMICO

Esta clase es usada para relacionar los puntos de la imagen con los distintos procesos del mundo exterior y sus valores de representación: posición en la pantalla, color de letra, color de fondo, etc. La rutina del Administrador de Mímicos se encarga de escribir en el pipe los valores del proceso que se está monitoreando, con una frecuencia definida para ese proceso.

Esta clase está compuesta, además, por dos clases de implementación: PIPE y MSG. Estas últimas sirven para establecer la comunicación entre el Administrador de Mímicos y el supervisor.

Los datos que componen esta relación son los siguientes:

- Segmento virtual de la imagen
- Índice del punto
- Índice del proceso
- Eje X de representación
- Eje Y de representación
- Color de letra
- Color de fondo
- Color de letra en estado de alarma
- Color de fondo en estado de alarma

Las funciones implementadas por esta clase son las siguientes:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar un mímico
- Borrar un mímico
- Leer un mímico
- Buscar los datos de representación dados un proceso y un punto de la imagen
- Administrar mímicos
- Lanzar el Administrador de Mímicos
- Detener el Administrador de Mímicos

2.3.1.10 PORT

Esta clase sirve para manejar los ports de comunicaciones a través de los cuales se intercambian datos entre IGNATIUS y el mundo exterior. También relaciona a los distintos dispositivos físicos con los puntos de la imagen asociados, permitiendo un esquema de relación de tipo muchos-a-muchos.

Fija los parámetros para establecer la comunicación con los ports (velocidad de transmisión, control de paridad, bits de datos, etc.). Lee la cola de salida del supervisor y escribe los mensajes en los ports asociados. Lee cada uno de los ports definidos a la espera de un mensaje, y cuando lo recibe lo escribe en la cola de entrada al supervisor.

Los datos que almacena para cada uno de los dispositivos físicos son:

- Dirección del port de comunicaciones (COM1, COM2,...)
- Dirección del dispositivo físico
- Manejador de archivo
- Descripción
- Fecha y hora de última actualización de la información

Los datos que almacena para cada una de las relaciones entre los dispositivos físicos y los puntos de la imagen son:

- Índice del dispositivo
- Segmento virtual de la imagen
- Índice del punto de la imagen

Las funciones asociadas son las siguientes:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar un dispositivo
- Modificar un dispositivo
- Leer un dispositivo
- Leer una relación
- Relacionar un dispositivo con un punto de la imagen
- Buscar el dispositivo relacionado a un punto de la imagen
- Borrar un dispositivo
- Borrar una relación
- Abrir un port
- Cerrar un port
- Leer un byte
- Escribir un byte
- Leer un string
- Escribir un string
- Leer la velocidad de transmisión
- Establecer la velocidad de transmisión
- Leer el estado del port
- Leer la cola de entrada del port
- Leer la cola de salida del port
- Leer la información del bloque de control de datos del port
- Establecer la información del bloque de control de datos del port
- Leer la información de la línea de control del port

- Establecer la información de la línea de control del port
- Establecer la información de control del modem del port
- Manejar los ports definidos
- Lanzar el Manejador de Ports
- Detener el Manejador de Ports

2.3.1.11 COMANDO

Esta clase provee al usuario un mecanismo para definir y administrar los comandos ingresados por el operador. Además posee un mecanismo de validación que permite determinar si un comando ingresado por el operador es válido y si los parámetros especificados son correctos.

En la arquitectura presentada, la definición, administración y validación de los comandos está separada de las rutinas de implementación de los mismos. Mientras que esta clase pretende encapsular las primeras funciones, es en el nivel de los Servicios Particulares donde se deben implementar las rutinas de atención de los mismos.

La ejecución de las rutinas de atención de los comandos puede ser llevada a cabo de dos formas: planificada o inmediata. En el primer caso, cada comando ingresado por el operador es agregado a una cola de comandos y atendido luego de haber atendido los comandos ingresados anteriormente. En el segundo caso, el comando ingresado es atendido primero, independientemente de los comandos que se encuentren encolados.

Los datos almacenados por esta clase para cada uno de los comandos definidos por el usuario son:

- Nombre del comando
- Dirección de la rutina de atención de comandos
- Máscara de edición de parámetros

Los datos almacenados por esta clase para cada uno de los comandos agregados a la cola de comandos son:

- Comando y parámetros
- Puntero usado para pasar datos a la rutina de atención de comandos
- Fecha y hora en que se agregó el comando a la cola
- Puntero al próximo elemento de la cola

Las funciones asociadas son las siguientes:

- Crear una instancia de la clase
- Destruir una instancia de la clase
- Agregar una definición de comando
- Borrar una definición de comando
- Leer una definición de comando
- Buscar la rutina de atención de comando de un comando
- Ejecutar una rutina de atención de comando en forma inmediata
- Agregar un comando a la cola de comandos
- Ejecutar cola de comandos (Intérprete de Comandos)
- Lanzar Intérprete de Comandos
- Detener Intérprete de Comandos
- Sacar un comando de la cola
- Verificar si existe un comando
- Validar parámetros ingresados

- Buscar un comando dentro de la lista de comandos

2.3.2 Jerarquía de las Clases

Dentro de la biblioteca, la clase TABLEAUX es heredada por las clases TABLEAUX_INT, TABLEAUX_DOUBLE y TABLEAUX_DIG, por una cuestión de implementación.

No se utilizaron las clases existentes en las bibliotecas UICL (User Interface Class Library) del VisualAge C++ para heredar de ellas atributos o métodos, ya que trató de conservarse lo más posible aspectos de portabilidad a otras plataformas de HW y SW.

Por supuesto, estas clases pueden ser usadas dentro de la programación del SCADA como base para derivar nuevas clases y modificar el comportamiento de determinados métodos que fueron definidos como virtuales con la intención de permitir redefiniciones en las clases herederas. Los métodos definidos como virtuales son los que están estrechamente relacionados con aspectos propios de implementación (sección crítica, threads, etc.), facilitando de esta manera la portabilidad de las clases.

2.3.3 Modelo de Datos

La siguiente figura representa el diagrama de entidad-relación del modelo de datos:

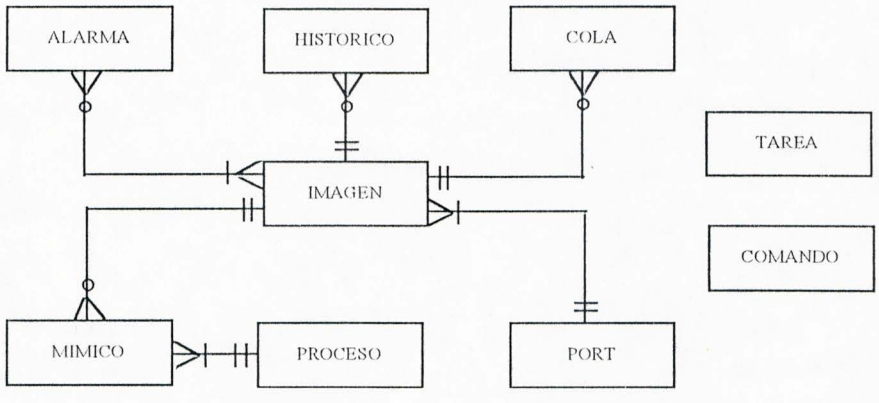


Figura 3. Modelo de Datos. Diagrama de Entidad-Relación

2.4 Descripción de la Implementación de IGNATIUS

IGNATIUS almacena internamente la información en arreglos de n elementos cuya dimensión la indica el usuario en el momento de crear el objeto, excepto en el caso de las Colas de Entrada y Salida que están representadas por listas encadenadas y el Histórico que se almacena en un archivo en disco.

Esto se debió a que en tiempo de programación, el programador que usa las clases conoce la cantidad de elementos que va a tener que representar, por ejemplo, sabe qué cantidad de alarmas se van a definir o qué cantidad de procesos se van a monitorear. Es preferible usar arreglos ya que se ahorra más espacio al usar estos en lugar de listas. Esto se debe a que las listas deben reservar para cada elemento, espacio para al menos un puntero al próximo elemento. Por otro lado la programación y el debugging son más sencillos usando arreglos.

La única excepción a esto es la clase Cola que es la que almacena los valores de los puntos de la imagen que van desde IGNATIUS al PLC y viceversa. Esta cantidad de elementos no se conoce en tiempo de programación. Por este motivo es que se usaron listas encadenadas en vez de arreglos.

La elección del uso de un archivo en disco para almacenar los valores del Histórico se debe a que la cantidad de información que se guarda en el Histórico depende de parámetros que no se conocen en tiempo de programación y que define el operador del sistema, como ser el porcentaje de actualización de un punto de la imagen que determina en qué circunstancia se debe grabar en el Histórico, por este motivo no se uso un arreglo. Tampoco se uso una lista encadenada ya que la cantidad de información que se puede almacenar en el histórico puede ser muy grande para almacenar en memoria RAM. Por otra parte es más seguro almacenar los registros históricos en un dispositivo de almacenamiento permanente y no temporario para evitar pérdida de registros ante fallas o caídas del sistema.

IGNATIUS está preparado para trabajar en ciclos de CPU 1/100 segundos, permitiendo planificar la ejecución de tareas de manera muy precisa.

Todos los registros de las tablas de IGNATIUS poseen un campo llamado timestamp destinado a almacenar la fecha y hora de última actualización del registro. Esto es a los efectos de poder determinar situaciones en las cuales hay datos que se vuelven críticos por no tener un tratamiento adecuado en el tiempo.

IGNATIUS se programó en OS/2, esto le da la capacidad de hacer procesamiento en paralelo. A continuación veremos algunos conceptos de el procesamiento en paralelo de OS/2 y como se aplican en IGNATIUS.

2.4.1 Capacidad de Procesamiento en Paralelo

La habilidad que tiene un SO para manejar la ejecución de más de una aplicación al mismo tiempo, se denomina procesamiento en paralelo si es que se cuenta con mas de un procesador, sino se denomina multitarea. Para el programador esto incluye la capacidad de hacer multiprogramación dentro de sus propios programas.

Veamos algunos conceptos del OS/2 relativos al concepto de multitarea extraídos de OS/2 Programming Guide [ORF92]:

Un *thread* es una unidad despachable de ejecución que consiste de un conjunto de instrucciones, valores relacionados de registros de CPU y un stack. Cada *proceso* tiene al menos un *thread*, llamado el *thread* principal o el *thread* 1, y puede tener varios *threads* ejecutando simultáneamente. La aplicación ejecuta cuando el sistema operativo le da el control al *thread* en el *proceso*. El *thread* es la unidad básica de ejecución.

Un *proceso* es el código, datos y otros recursos, como file handlers, semáforos, pipes, colas, etc., de una aplicación en memoria. El OS/2 considera a cada aplicación que carga para su ejecución como a un *proceso*. Los recursos del sistema son asignados a nivel *proceso*.

Una *sesión* es uno o más *procesos* , cada uno de ellos con su consola virtual (una consola virtual es una pantalla virtual, ya sea basada en caracter, pantalla completa o pantalla Presentation Manager y los buflèrs de entrada para pantalla y teclado).

El sistema operativo OS/2 soporta hasta 255 *sesiones* concurrentes y hasta 4095 *procesos*. El SO soporta en total hasta un máximo de 4095 *threads*., aunque el número de *threads* en un único *proceso* será menor y variará de acuerdo al uso de recursos dentro del *proceso*.

Luego de haber introducido estos conceptos, se puede clasificar el nivel de multiprocesamiento de la herramienta de acuerdo a la siguiente figura:

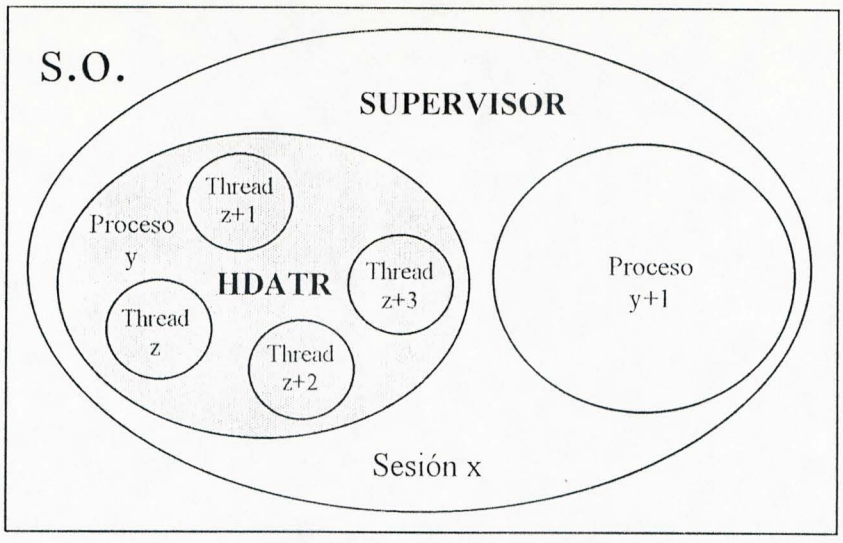


Figura 2. Multiprocesamiento. Sesiones, Procesos y Threads.

IGNATIUS se encuentra posicionado a nivel proceso. Provee el código y los datos como así también distintos recursos como ser pipes, semáforos y file handles.. Dentro de este proceso principal, provee un thread principal y un thread para cada una de las tareas del motor de ejecución.

La aplicación SCADA, que usa IGNATIUS, se posiciona a nivel sesión, proveyendo la consola virtual de la aplicación.

2.4.2 Especificación de las Clases

2.4.2.1 CLASE tableaux

La imagen se encuentra dividida virtualmente en tres segmentos de acuerdo a los distintos tipos de datos que en ella se pueden almacenar (digitales, analógicos y enteros). La clase abstracta *tableaux*, junto con las clases *tableaux_int, tableaux_double, tableaux_dig* y la jerarquía que ellas representan conforman la implementación de la imagen del sistema. En esta jerarquía de clases, *tableaux* es la clase heredada por las clases *tableaux_int, tableaux_double y tableaux_dig*.

Si bien la imagen se encuentra virtualmente dividida de acuerdo al tipo de datos, todos los puntos de la imagen se manejan dentro de IGNATIUS de la misma manera y reciben el mismo tratamiento (por ej.: un punto de la imagen se almacena en el archivo histórico de una única manera independientemente del tipo de datos que representa). La clase abstracta *tableaux* provee una referencia única para todos los puntos de la imagen independientemente del tipo de los datos.

La siguiente es la definición de la clase *tableaux* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class tableaux {
public:
   virtual int agregar_punto(double valor,double valor_set_point,short estado,short cant_ref,short porc_act) = 0;
   virtual int modificar_punto(int indice,double valor,double valor_set_point,short estado,short cant_ref,
                               short porc_act) = 0;
   virtual int borrar_punto(int indice) = 0;
   virtual int leer_punto(int indice,double& valor,double& valor_set_point,short& estado,short& cant_ref,
```

```
                                short& porc_act) = 0;
    virtual int modificar_valor(int indice, double valor) = 0;
    virtual int sumar_uno_cant_ref(int indice) = 0;
    virtual int modificar_estado(int indice,short estado) = 0;
    virtual int cant_puntos() = 0;
    virtual short tipo_de_tableaux() = 0;
    virtual int ack_alm(int indice) = 0;
protected:
    virtual unsigned long EntrarCritSec() = 0;
    virtual unsigned long SalirCritSec() = 0;
};
```

Por ser una clase abstracta no existe implementación para los métodos.

2.4.2.2 CLASE tableaux_int

La clase *tableaux_int* representa la porción de la imagen del sistema cuyos elementos son enteros. Es derivada de la clase abstracta *tableaux*.

La siguiente es la definición de la clase *tableaux_int* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class tableaux_int : public tableaux{
    struct reg_tableaux {
        int valor;
        int valor_set_point;
        short estado;
        short cant_ref;
        short porc_act;
        char timestamp[20];
    };
    reg_tableaux* v;
    int cant_regs, long_tabla;
public:
    tableaux_int(int s=50);
    ~tableaux_int();
    int agregar_punto(double valor,double valor_set_point,short estado,short cant_ref,short porc_act);
    int modificar_punto(int indice,double valor,double valor_set_point,short estado,short cant_ref,short porc_act);
    int borrar_punto(int indice);
    int leer_punto(int indice,double& valor,double& valor_set_point,short& estado,short& cant_ref,short& porc_act);
    int modificar_valor(int indice,double valor);
    int sumar_uno_cant_ref(int indice);
    int modificar_estado(int indice,short estado);
    int cant_puntos();
    short tipo_de_tableaux();
    int ack_alm(int indice);
protected:
    virtual unsigned long EntrarCritSec();
    virtual unsigned long SalirCritSec();
};
```

La clase *tableaux_int* posee los siguientes **atributos**:

1. REG_TABLEAUX:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* TABLEAUX_INT
- *Descripción:* representa la estructura de un punto de la imagen.

1.1 VALOR:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado

- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_TABLEAUX
- *Descripción:* representa el valor de un punto de la imagen en un instante determinado.

1.2 VALOR_SET_POINT:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_TABLEAUX
- *Descripción:* representa el valor de set point de un punto de la imagen.

1.3 ESTADO:
- *Tipo de Datos:* short
- *Tipo de Acceso:* privado
- *Rango* -1..8
  -1: error
  0: ok
  1: alarma digital
  2: alarma por baja
  3: alarma por alta
  4: alarma por tasa de cambios
  5: alarma digital reconocida
  6: alarma por baja reconocida
  7: alarma por alta reconocida
  8: alarma por tasa de cambios reconocida
- *Parte de:* REG_TABLEAUX
- *Descripción:* representa el estado de un punto de la imagen en un instante determinado.

1.4 CANT_REF:
- *Tipo de Datos:* short
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_TABLEAUX
- *Descripción:* representa la cantidad de veces que sufrió alguna modificación un punto de la imagen.

1.5 PORC_ACT:
- *Tipo de Datos:* short
- *Tipo de Acceso:* privado
- *Rango:* 0..100
- *Parte de:* REG_TABLEAUX
- *Descripción:* representa el porcentaje de modificación del valor de un punto con respecto al valor anterior, a partir del cual se desea registrar el cambio en el archivo histórico.

1.6 TIMESTAMP:
- *Tipo de Datos:* caracter de 20 posiciones
- *Tipo de Acceso:* privado
- *Rango:* cualquier fecha representable en el sistema con el siguiente formato: aaaa-mm-dd.hh:mm:ss

  donde:

  aaaa = año
  mm = mes
  dd = día
  hh = hora
  mm = minutos
  ss = segundos

- *Parte de:* REG_TABLEAUX
- *Descripción:* representa la fecha y hora en la cual fue modificado por última vez algún valor de un punto de la imagen

2. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_TABLEAUX
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_INT
- *Descripción:* cuando se crea un objeto se reserva espacio para n puntos de la imagen (elementos de tipo REG_TABLEAUX). V es el puntero a este espacio de memoria.

3. CANT_REGS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_INT
- *Descripción:* representa la cantidad de puntos de la imagen almacenados en el objeto.

4. LONG_TABLA
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_INT
- *Descripción:* representa la cantidad máxima de puntos de la imagen que pueden ser almacenados por el objeto.

La clase *tableaux_int* posee los siguientes **métodos**:

1. TABLEAUX_INT (constructor):
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* es el constructor de la clase. Reserva espacio para almacenar los puntos de la imagen e inicializa los valores de los atributos. El espacio reservado para los puntos de la imagen es alocado dinámicamente con el formato de un arreglo de *s* elementos de tipo REG_TABLEAUX. Se puede invocar pasando como parámetro un número entero *s* para indicar la cantidad de puntos de la imagen que va a almacenar el objeto como máximo; en caso contrario, se reserva espacio para un máximo de 50 puntos de la imagen de tipo entero.
- *Devuelve:* no devuelve ningún valor
- *Parámetros:*

  s:
  - *E/S:* entrada
  - *Tipo de Datos:* entero
  - *Rango:* cualquier valor positivo representable por el tipo de datos
  - *Descripción:* se utiliza para indicar la cantidad de puntos de la imagen que debe almacenar el objeto como máximo.

2. ~TABLEAUX_INT (destructor):
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* es el destructor de la clase. Libera el espacio alocado por el constructor del objeto.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_PUNTO:

- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* permite agregar un punto de la imagen.
- *Devuelve:* un valor entero positivo que indica la posición donde se agregó el punto, o bien -1 en caso de que no se hubiera podido agregar por falta de espacio.
- *Parámetros:*

valor:
- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el valor del punto a agregar.

valor_set_point:
- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el valor de set point del punto a agregar.

estado:
- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* cualquier valor representable por el tipo de datos (ver Atributos)
- *Descripción:* representa el estado del punto a agregar.

cant_ref:
- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la cantidad de modificaciones que sufrió el punto a agregar en alguno de sus atributos (inicialmente debería ser igual a 0).

porc_act:
- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* 0..100
- *Descripción:* representa el porcentaje de modificación del valor con respecto al valor anterior, a partir del cual se desea registrar el cambio en el archivo histórico para el punto a agregar.

4. MODIFICAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* permite modificar un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:*

índice:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* algún valor que represente el índice de un punto existente
- *Descripción:* representa el valor del índice del punto a modificar.

valor:
- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el nuevo valor del punto a modificar.

valor_set_point:
- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el nuevo valor de set point del punto a modificar.

estado:
- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* cualquier valor representable por el tipo de datos (ver Atributos)
- *Descripción:* representa el nuevo estado del punto a modificar.

cant_ref:
- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la nueva cantidad de modificaciones que sufrió el punto en alguno de sus atributos.

porc_act:
- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* 0..100
- *Descripción:* representa el nuevo porcentaje de modificación del valor con respecto al valor anterior, a partir del cual se desea registrar el cambio en el archivo histórico.

5. BORRAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* permite borrar un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:*

índice:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* algún valor que represente el índice de un punto existente
- *Descripción:* representa el valor del índice del punto a borra.

6. LEER_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* permite leer los valores de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:*

índice:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* algún valor que represente el índice de un punto existente
- *Descripción:* representa el valor del índice del punto a leer.

valor:
- *E/S:* salida
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el valor del punto leído.

valor_set_point:
- *E/S:* salida
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el valor de set point del punto leído.

estado:
- *E/S:* salida
- *Tipo de Datos:* short
- *Rango:* cualquier valor representable por el tipo de datos (ver Atributos)

- *Descripción:* representa el estado del punto leído.

cant_ref:

- *E/S:* salida
- *Tipo de Datos:* short
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la cantidad de modificaciones que sufrió el punto leído en alguno de sus atributos.

porc_act:

- *E/S:* salida
- *Tipo de Datos:* short
- *Rango:* 0..100
- *Descripción:* representa el porcentaje de modificación del valor con respecto al valor anterior, a partir del cual se registra el cambio en el archivo histórico para el punto leído.

7. MODIFICAR_VALOR:

- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* permite modificar el valor de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:*

índice:

- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* algún valor que represente el índice de un punto existente
- *Descripción:* representa el valor del índice del punto a modificar.

valor:

- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el nuevo valor del punto a modificar.

8. SUMAR_UNO_CANT_REF:

- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* suma uno al atributo *cant_ref* de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:*

índice:

- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* algún valor que represente el índice de un punto existente
- *Descripción:* representa el valor del índice del punto a modificar.

9. MODIFICAR_ESTADO:

- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* permite modificar el estado de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:*

índice:

- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* algún valor que represente el índice de un punto existente
- *Descripción:* representa el valor del índice del punto a modificar.

estado:

- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* cualquier valor representable por el tipo de datos (ver Atributos)
- *Descripción:* representa el valor del nuevo estado.

10. CANT_PUNTOS:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* devuelve la cantidad de puntos de la imagen almacenados por el objeto.
- *Devuelve: cant_regs..*

11. TIPO_DE_TABLEAUX:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* devuelve el tipo del tableaux (entero).
- *Devuelve:* 0.

12. ACK_ALM:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_INT
- *Descripción:* cambia el estado de alarma a estado de alarma reconocido para un punto de la imagen determinado.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:*

índice:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* algún valor que represente el índice de un punto existente
- *Descripción:* representa el valor del índice del punto a reconocer

13 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* TABLEAUX_INT
- *Descripción:* marca el comienzo de una sección crítica. Es usado dentro de los otros métodos de la clase para manejar el acceso concurrente a secciones críticas. Fue definido como *virtual* para permitir la redefinición por cuestiones de implementación (encapsula una sentencia del Programa de Control del Sistema Operativo)
- *Devuelve:*

0	NO_ERROR
309	ERROR_INVALID_THREADID
484	ERROR_CRITSEC_OVERFLOW

- *Parámetros*: no tiene.

14 SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* TABLEAUX_INT
- *Descripción:* marca el fin de una sección crítica. Es usado dentro de los otros métodos de la clase para manejar el acceso concurrente a secciones críticas. Fue definido como *virtual* para permitir la redefinición por cuestiones de implementación (encapsula una sentencia del Programa de Control del Sistema Operativo)
- *Devuelve:*

0	NO_ERROR
309	ERROR_INVALID_THREADID
485	ERROR_CRITSEC_UNDERFLOW

- *Parámetros*: no tiene.

2.4.2.3 CLASE tableaux_double

La clase *tableaux_double* representa la porción de la imagen del sistema cuyos elementos son analógicos. Es derivada de la clase abstracta *tableaux*.

La siguiente es la definición de la clase *tableaux_double* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class tableaux_double : public tableaux{
   struct reg_tableaux {
      double valor;
      double valor_set_point;
      short estado;
      short cant_ref;
      short porc_act;
      char timestamp[20];
   };
   reg_tableaux* v;
   int cant_regs, long_tabla;
public:
   tableaux_double(int s=50);
   ~tableaux_double();
   int agregar_punto(double valor,double valor_set_point,short estado,short cant_ref,short porc_act);
   int modificar_punto(int indice,double valor,double valor_set_point,short estado,short cant_ref,short porc_act);
   int borrar_punto(int indice);
   int leer_punto(int indice,double& valor,double& valor_set_point,short& estado,short& cant_ref,short& porc_act);
   int modificar_valor(int indice,double valor);
   int sumar_uno_cant_ref(int indice);
   int modificar_estado(int indice,short estado);
   int cant_puntos();
   short tipo_de_tableaux();
   int ack_alm(int indice);
protected:
   virtual unsigned long EntrarCritSec();
   virtual unsigned long SalirCritSec();
};
```

La clase *tableaux_double* posee los siguientes **atributos**:

1. REG_TABLEAUX:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* representa la estructura de un punto de la imagen.

1.1 VALOR:
- *Tipo de Datos:* double
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_TABLEAUX
- *Descripción:* representa el valor de un punto de la imagen en un instante determinado.

1.2 VALOR_SET_POINT:
- *Tipo de Datos:* double
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_TABLEAUX
- *Descripción:* representa el valor de set point de un punto de la imagen.

1.3 ESTADO: idem *tableaux_int*

1.4 CANT_REF: idem *tableaux_int*

1.5 PORC_ACT: idem *tableaux_int*

1.6 TIMESTAMP: idem *tableaux_int*

2. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_TABLEAUX
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* cuando se crea un objeto se reserva espacio para n puntos de la imagen (elementos de tipo REG_TABLEAUX). V es el puntero a este espacio de memoria.

3. CANT_REGS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* representa la cantidad de puntos de la imagen almacenados en el objeto.

4. LONG_TABLA
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* representa la cantidad máxima de puntos de la imagen que pueden ser almacenados por el objeto.

La clase *tableaux_double* posee los siguientes **métodos**:

1. TABLEAUX_DOUBLE (constructor):
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* es el constructor de la clase. Reserva espacio para almacenar los puntos de la imagen e inicializa los valores de los atributos. Se puede invocar pasando como parámetro un número entero para indicar la cantidad de puntos de la imagen que va a almacenar el objeto como máximo; en caso contrario, se reserva espacio para un máximo de 50 puntos de la imagen de tipo double.
- *Devuelve:* no devuelve ningún valor
- *Parámetros:* idem *tableaux_int*

2. ~TABLEAUX_DOUBLE (destructor):
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* es el destructor de la clase. Libera el espacio alocado por el constructor del objeto.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* permite agregar un punto de la imagen.

- *Devuelve:* un valor entero positivo que indica la posición donde se agregó el punto, o bien -1 en caso de que no se hubiera podido agregar por falta de espacio.
- *Parámetros*: idem *tableaux_int*

4. MODIFICAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* permite modificar un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

5. BORRAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* permite borrar un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

6. LEER_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* permite leer los valores de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros*: idem *tableaux_int*

7. MODIFICAR_VALOR:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* permite modificar el valor de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

8. SUMAR_UNO_CANT_REF:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* suma uno al atributo *cant_ref* de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

9. MODIFICAR_ESTADO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* permite modificar el estado de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

10. CANT_PUNTOS:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* devuelve la cantidad de puntos de la imagen almacenados por el objeto.
- *Devuelve: cant_regs..*

11. TIPO_DE_TABLEAUX:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE

- *Descripción:* devuelve el tipo del tableaux (double).
- *Devuelve:* 1.

12. ACK_ALM:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* cambia el estado de alarma a estado de alarma reconocido para un punto de la imagen determinado.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

13 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* marca el comienzo de una sección crítica. Es usado dentro de los otros métodos de la clase para manejar el acceso concurrente a secciones críticas. Fue definido como *virtual* para permitir la redefinición por cuestiones de implementación (encapsula una sentencia del Programa de Control del Sistema Operativo)
- *Devuelve:* idem *tableaux_int*
- *Parámetros*: no tiene.

14 SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* TABLEAUX_DOUBLE
- *Descripción:* marca el fin de una sección crítica. Es usado dentro de los otros métodos de la clase para manejar el acceso concurrente a secciones críticas. Fue definido como *virtual* para permitir la redefinición por cuestiones de implementación (encapsula una sentencia del Programa de Control del Sistema Operativo)
- *Devuelve:* idem *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.4 CLASE tableaux_dig

La clase *tableaux_dig* representa la porción de la imagen del sistema cuyos elementos son digitales. Es derivada de la clase abstracta *tableaux*.

La siguiente es la definición de la clase *tableaux_dig* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class tableaux_dig : public tableaux{
   struct reg_tableaux {
      int valor;
      int valor_set_point;
      short estado;
      short cant_ref;
      short porc_act;
      char timestamp[20];
   };
   reg_tableaux* v;
   int cant_regs, long_tabla;
public:
   tableaux_dig(int s=50);
   ~tableaux_dig();
   int agregar_punto(double valor,double valor_set_point,short estado,short cant_ref,short porc_act);
   int modificar_punto(int indice,double valor,double valor_set_point,short estado,short cant_ref,short porc_act);
   int borrar_punto(int indice);
```

```
    int leer_punto(int indice,double& valor,double& valor_set_point,short& estado,short& cant_ref,short& porc_act);
    int modificar_valor(int indice,double valor);
    int sumar_uno_cant_ref(int indice);
    int modificar_estado(int indice,short estado);
    int cant_puntos();
    short tipo_de_tableaux();
    int ack_alm(int indice);
protected:
    virtual unsigned long EntrarCritSec();
    virtual unsigned long SalirCritSec();
};
```

La clase *tableaux_dig* posee los siguientes **atributos**:

1. REG_TABLEAUX:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* TABLEAUX_DIG
- *Descripción:* representa la estructura de un punto de la imagen.

1.1 VALOR: idem *tableaux_int*

1.2 VALOR_SET_POINT: idem *tableaux_int*

1.3 ESTADO: idem *tableaux_int*

1.4 CANT_REF: idem *tableaux_int*

1.5 PORC_ACT: idem *tableaux_int*

1.6 TIMESTAMP: idem *tableaux_int*

2. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_TABLEAUX
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_DIG
- *Descripción:* cuando se crea un objeto se reserva espacio para n puntos de la imagen (elementos de tipo REG_TABLEAUX). V es el puntero a este espacio de memoria.

3. CANT_REGS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_DIG
- *Descripción:* representa la cantidad de puntos de la imagen almacenados en el objeto.

4. LONG_TABLA
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TABLEAUX_DIG
- *Descripción:* representa la cantidad máxima de puntos de la imagen que pueden ser almacenados por el objeto.

La clase *tableaux_dig* posee los siguientes **métodos**:

1. TABLEAUX_DIG (constructor):
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* es el constructor de la clase. Reserva espacio para almacenar los puntos de la imagen e inicializa los valores de los atributos. Se puede invocar pasando como parámetro un número entero para indicar la cantidad de puntos de la imagen que va a almacenar el objeto como máximo; en caso contrario, se reserva espacio para un máximo de 50 puntos de la imagen de tipo entero.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*: idem *tableaux_int*

2. ~TABLEAUX_DIG (destructor):
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* es el destructor de la clase. Libera el espacio alocado por el constructor del objeto.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* permite agregar un punto de la imagen.
- *Devuelve:* un valor entero positivo que indica la posición donde se agregó el punto, o bien -1 en caso de que no se hubiera podido agregar por falta de espacio.
- *Parámetros*: idem *tableaux_int*

4. MODIFICAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* permite modificar un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

5. BORRAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* permite borrar un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

6. LEER_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* permite leer los valores de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros*: idem *tableaux_int*

7. MODIFICAR_VALOR:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* permite modificar el valor de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

8. SUMAR_UNO_CANT_REF:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* suma uno al atributo *cant_ref* de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

9. MODIFICAR_ESTADO:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* permite modificar el estado de un punto de la imagen existente.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

10. CANT_PUNTOS:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* devuelve la cantidad de puntos de la imagen almacenados por el objeto.
- *Devuelve: cant_regs..*

11. TIPO_DE_TABLEAUX:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* devuelve el tipo del tableaux (digital).
- *Devuelve:* 2.

12. ACK_ALM:
- *Tipo de Acceso:* público
- *Parte de:* TABLEAUX_DIG
- *Descripción:* cambia el estado de alarma a estado de alarma reconocido para un punto de la imagen determinado.
- *Devuelve:* 0 si se pudo realizar la operación y -1 en caso contrario.
- *Parámetros:* idem *tableaux_int*

13 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* TABLEAUX_DIG
- *Descripción:* marca el comienzo de una sección crítica. Es usado dentro de los otros métodos de la clase para manejar el acceso concurrente a secciones críticas. Fue definido como *virtual* para permitir la redefinición por cuestiones de implementación (encapsula una sentencia del Programa de Control del Sistema Operativo)
- *Devuelve:* idem *tableaux_int*
- *Parámetros*: no tiene.

14 SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* TABLEAUX_DIG
- *Descripción:* marca el fin de una sección crítica. Es usado dentro de los otros métodos de la clase para manejar el acceso concurrente a secciones críticas. Fue definido como *virtual* para permitir la redefinición por cuestiones de implementación (encapsula una sentencia del Programa de Control del Sistema Operativo)
- *Devuelve:* idem *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.5 CLASE alarma

La clase *alarma* representa todas las condiciones de alarma del sistema y su relación con los distintos puntos de la imagen.

La siguiente es la definición de la clase *alarma* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class alarma {
  struct reg_alarma {
    char nombre[12];
    int prioridad;
    PFNTHREAD proc_usu;
    double al_min, al_max;
    int criterio, cant_veces;
    unsigned long int frecuencia, tpo_esp;
    long tasa_max,tasa_min;
    char timestamp[20];
  };
  struct reg_alarma_imagen {
    int almid, indice;
    tableaux* tabla;
  };
  reg_alarma* v;
  reg_alarma_imagen* w;
  Boolean continuar;
  IWindow* pVentana;
public:
  int cant_alarmas, cant_relac, long_tabla;
  alarma(int s=10);
  ~alarma();
  int agregar_alarma(char* nombre,unsigned long int frecuencia,int prioridad,PFNTHREAD proc_usu,int criterio,double
                     al_min,double al_max,unsigned long int tpo_esp,int cant_veces,long tasa_max,long tasa_min);
  int modificar_alarma(int almid,char* nombre,unsigned long int frecuencia,int prioridad,PFNTHREAD proc_usu,int
                       criterio,double al_min,double al_max,unsigned long int tpo_esp,int cant_veces,long tasa_max,
                       long tasa_min);
  int leer_alarma(int almid,char* nombre,unsigned long int& frecuencia,int& prioridad,PFNTHREAD& proc_usu,int&
                  criterio,double& al_min,double& al_max,unsigned long int& tpo_esp,int& cant_veces,
                  long& tasa_max,long& tasa_min);
  int sumar_uno_alarma(int almid);
  int calcular_min_max(int almid,double set_point,int porc_tolerancia);
  int buscar_alarma(tableaux* tabla,int indice);
  int buscar_punto(int almid,tableaux*& tabla,int& indice);
  int relacionar_alarma_imagen(int almid,tableaux* tabla,int indice);
  int borrar_alarma(int almid);
  int borrar_relac(int relid);
  int leer_relac(int relid,int& almid,tableaux*& tabla,int& indice);
  void mostrar_alarmas();
  void mostrar_relac();
  void arrancar();
  virtual IThread* lanzar_alarma(IEV& semaforo,IWindow* p_ventana);
  virtual void detener_alarma();
protected:
  virtual unsigned long EntrarCritSec();
  virtual unsigned long SalirCritSec();
};
```

La clase *alarma* posee los siguientes **atributos**:

1. REG_ALARMA:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* ALARMA
- *Descripción:* representa la estructura de una condición de alarma.

- *Parte de:* REG_ALARMA
- *Descripción:* representa el criterio con el cual el motor de ejecución de ALARMA evalúa la condición de alarma. Por ej.: si el criterio es ALTA, el motor de ejecución de ALARMA evaluará si el valor de el/los punto/s de la imagen relacionado/s supera/n el valor especificado en AL_MAX; en caso afirmativo, el/los punto/s relacionado/s entrará/n en condición de alarma y se lanzará la rutina de atención de alarma especificada en PROC_USU.

1.7 CANT_VECES:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* REG_ALARMA
- *Descripción:* representa la cantidad de veces que el motor de ejecución de ALARMA evaluó la condición de alarma en función de la frecuencia. Este atributo se provee para fines informativos o estadísticos.

1.8 FRECUENCIA:
- *Tipo de Datos:* unsigned long int
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_ALARMA
- *Descripción:* representa la frecuencia de tiempo con la que el motor de ejecución de ALARMA debe evaluar la condición de alarma. Este valor debe ser expresado en centésimas de segundo (ver CICLOS DE CPU).

1.9 TPO_ESP:
- *Tipo de Datos:* unsigned long int
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_ALARMA
- *Descripción:* representa el tiempo transcurrido, expresado en centésimas de segundo, desde la última evaluación hecha por el motor de ejecución de ALARMA.

1.10 TASA_MAX:
- *Tipo de Datos:* long
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* REG_ALARMA
- *Descripción:* representa la cantidad máxima de cambios que puede sufrir un punto de la imagen sin entrar en estado de alarma Este valor es evaluado por el motor de ejecución de ALARMA comparándolo con la cantidad de referencias (CANT_REF).

1.11 TASA_MIN:
- *Tipo de Datos:* long
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* REG_ALARMA
- *Descripción:* representa la cantidad mínima de cambios que puede sufrir un punto de la imagen sin entrar en estado de alarma Este valor es evaluado por el motor de ejecución de ALARMA comparándolo con la cantidad de referencias (CANT_REF).

1.12 TIMESTAMP:
- *Tipo de Datos:* caracter de 20 posiciones
- *Tipo de Acceso:* privado

- *Rango:* cualquier fecha representable en el sistema con el siguiente formato: aaaa-mm-dd.hh:mm:ss

donde:

aaaa = año
mm = mes
dd = día
hh = hora
mm = minutos
ss = segundos

- *Parte de:* ALARMA
- *Descripción:* representa la fecha y hora en la cual fue modificado por última vez algún valor de una condición de alarma.

2. REG_ALARMA_IMAGEN:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* ALARMA
- *Descripción:* representa la relación entre las condiciones de alarma y los puntos de la imagen. Implementa una relación de muchos a muchos.

2.1 ALMID:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_ALARMA_IMAGEN
- *Descripción:* representa el índice de la condición de alarma relacionada.

2.2 INDICE:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* REG_ALARMA_IMAGEN
- *Descripción:* representa el índice del punto de la imagen relacionado.

2.3 TABLA:
- *Tipo de Datos:* puntero a TABLEAUX
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_ALARMA_IMAGEN
- *Descripción:* representa el puntero al segmento virtual de la imagen relacionado.

3. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_ALARMA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* ALARMA
- *Descripción:* cuando se crea un objeto se reserva espacio para n condiciones de alarma (elementos de tipo REG_ALARMA). V es el puntero a este espacio de memoria.

4. W:
- *Tipo de Datos:* puntero a estructura de tipo REG_ALARMA_IMAGEN
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos

- *Parte de:* ALARMA
- *Descripción:* cuando se crea un objeto se reserva espacio para n relaciones alarma_imagen (elementos de tipo REG_ALARMA_IMAGEN). W es el puntero a este espacio de memoria.

5. CONTINUAR:
- *Tipo de Datos:* boolean
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* ALARMA
- *Descripción:* es el atributo evaluado por el motor de ejecución de ALARMA para determinar si debe continuar o debe finalizar su ejecución.

6. PVENTANA:
- *Tipo de Datos:* puntero a tipo IWindow (UICL)
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* ALARMA
- *Descripción:* es un puntero a una ventana de tipo IWindow (UICL) del Presentation Manager. Este valor no es usado por IGNATIUS, sólo se usa para pasar como parámetro a la rutina de atención de alarma escrita por el usuario; esto es útil ya que si una rutina de atención de alarma usa ventanas de PM, necesita conocer cuál es la ventana que se encuentra más arriba en la jerarquía de ventanas de la aplicación (ventana owner) [ver OS/2 Presentation Manager Guide and Reference].

7. CANT_ALARMAS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* ALARMA
- *Descripción:* representa la cantidad de condiciones de alarma existentes en el objeto.

8. CANT_RELAC:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* ALARMA
- *Descripción:* representa la cantidad de relaciones alarma-imagen existentes en el objeto.

9. LONG_TABLA:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* ALARMA
- *Descripción:* representa la cantidad máxima de condiciones de alarma que puede almacenar el objeto.

La clase *alarma* posee los siguientes **métodos**:

1. ALARMA (constructor):
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* es el constructor de la clase. Reserva espacio para almacenar las condiciones de alarma y sus relaciones con los puntos de la imagen; también inicializa los valores de los atributos. El espacio reservado para las condiciones de alarma es alocado dinámicamente con el formato de un arreglo de *s* elementos de tipo REG_ALARMA. El espacio reservado para las relaciones es alocado dinámicamente con el formato de un arreglo de *s*3* elementos de tipo REG_ALARMA_IMAGEN. Se

puede invocar pasando como parámetro un número entero *s* para indicar la cantidad de condiciones de alarma y relaciones que el objeto va a almacenar como máximo; en caso contrario, se reserva espacio para un máximo de 10 condiciones de alarma y 30 relaciones.
- *Devuelve:* no devuelve ningún valor
- *Parámetros:*

s:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Descripción:* se utiliza para indicar la cantidad de condiciones de alarma y relaciones que debe almacenar el objeto como máximo.

2. ~ALARMA (destructor):
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* es el destructor de la clase. Libera el espacio alocado por el constructor del objeto.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_ALARMA:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* si hay espacio, agrega una condición de alarma en el arreglo.
- *Devuelve:* un valor entero positivo que indica la posición del arreglo donde se agregó la condición, o bien -1 en caso de que no se hubiera podido agregar por falta de espacio.
- *Parámetros:*

nombre:
- *E/S:* entrada
- *Tipo de Datos:* puntero a caracter.
- *Rango:* ver NOMBRE en *Atributos*.
- *Descripción:* ver NOMBRE en *Atributos*.

frecuencia
- *E/S:* entrada
- *Tipo de Datos:* unsigned long int
- *Rango:* ver FRECUENCIA en *Atributos*.
- *Descripción:* ver FRECUENCIA en *Atributos*.

prioridad:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* ver PRIORIDAD en *Atributos*.
- *Descripción:* ver PRIORIDAD en *Atributos*.

proc_usu:
- *E/S:* entrada
- *Tipo de Datos:* PFNTHREAD
- *Rango:* ver PROC_USU en *Atributos*.
- *Descripción:* ver PROC_USU en *Atributos*.

criterio:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* ver CRITERIO en *Atributos*.
- *Descripción:* ver CRITERIO en *Atributos*.

al_min:
- *E/S:* entrada
- *Tipo de Datos:* double

- *Rango:* ver AL_MIN en *Atributos.*
- *Descripción:* ver AL_MIN en *Atributos.*

al_max:
- *E/S:* entrada
- *Tipo de **Datos**:* double
- *Rango:* ver AL_MAX en *Atributos.*
- *Descripción:* ver AL_MAX en *Atributos.*

tpo_esp:
- *E/S:* entrada
- *Tipo de Datos:* unsigned long int
- *Rango:* ver TPO_ESP en *Atributos.*
- *Descripción:* ver TPO_ESP en *Atributos.*

cant_veces:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* ver CANT_VECES en *Atributos.*
- *Descripción:* ver CANT_VECES en *Atributos.*

tasa_max:
- *E/S:* entrada
- *Tipo de Datos:* long
- *Rango:* ver TASA_MAX en *Atributos.*
- *Descripción:* ver TASA_MAX en *Atributos.*

tasa_min:
- *E/S:* entrada
- *Tipo de Datos:* long
- *Rango:* ver TASA_MIN en *Atributos.*
- *Descripción:* ver TASA_MIN en *Atributos.*

4. MODIFICAR_ALARMA:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* si el índice especificado es válido, modifica la condición de alarma con los nuevos valores especificados.
- *Devuelve:* 0 si pudo modificar, -1 si no pudo modificar.
- *Parámetros*

almid:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la posición dentro del arreglo de la condición de alarma a modificar.

nombre: idem *AGREGAR_ALARMA*
frecuencia: idem *AGREGAR_ALARMA*
prioridad: idem *AGREGAR_ALARMA*
proc_usu: idem *AGREGAR_ALARMA*
criterio: idem *AGREGAR_ALARMA*
al_min: idem *AGREGAR_ALARMA*
al_max: idem *AGREGAR_ALARMA*
tpo_esp: idem *AGREGAR_ALARMA*
cant_veces: idem *AGREGAR_ALARMA*
tasa_max: idem *AGREGAR_ALARMA*
tasa_min: idem *AGREGAR_ALARMA*

5. LEER_ALARMA:
- *Tipo de Acceso:* público

- *Parte de:* ALARMA
- *Descripción:* devuelve los valores de una condición de alarma especificada.
- *Devuelve:* si la operación termina bien devuelve 0, si termina mal devuelve -1.
- *Parámetros*

almid:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la posición dentro del arreglo de la condición de alarma a leer.

nombre:
- *E/S:* salida
- *Tipo de Datos:* puntero a caracter.
- *Rango:* ver NOMBRE en *Atributos*.
- *Descripción:* ver NOMBRE en *Atributos*.

frecuencia
- *E/S:* salida
- *Tipo de Datos:* unsigned long int
- *Rango:* ver FRECUENCIA en *Atributos*.
- *Descripción:* ver FRECUENCIA en *Atributos*.

prioridad:
- *E/S:* salida
- *Tipo de Datos:* entero
- *Rango:* ver PRIORIDAD en *Atributos*.
- *Descripción:* ver PRIORIDAD en *Atributos*.

proc_usu:
- *E/S:* salida
- *Tipo de Datos:* PFNTHREAD
- *Rango:* ver PROC_USU en *Atributos*.
- *Descripción:* ver PROC_USU en *Atributos*.

criterio:
- *E/S:* salida
- *Tipo de Datos:* entero
- *Rango:* ver CRITERIO en *Atributos*.
- *Descripción:* ver CRITERIO en *Atributos*.

al_min:
- *E/S:* salida
- *Tipo de Datos:* double
- *Rango:* ver AL_MIN en *Atributos*.
- *Descripción:* ver AL_MIN en *Atributos*.

al_max:
- *E/S:* salida
- *Tipo de Datos:* double
- *Rango:* ver AL_MAX en *Atributos*.
- *Descripción:* ver AL_MAX en *Atributos*.

tpo_esp:
- *E/S:* salida
- *Tipo de Datos:* unsigned long int
- *Rango:* ver TPO_ESP en *Atributos*.
- *Descripción:* ver TPO_ESP en *Atributos*.

cant_veces:
- *E/S:* salida
- *Tipo de Datos:* entero
- *Rango:* ver CANT_VECES en *Atributos*.
- *Descripción:* ver CANT_VECES en *Atributos*.

tasa_max:

- *E/S:* salida
- *Tipo de Datos:* long
- *Rango:* ver TASA_MAX en *Atributos*.
- *Descripción:* ver TASA_MAX en *Atributos*.

tasa_min:
- *E/S:* salida
- *Tipo de Datos:* long
- *Rango:* ver TASA_MIN en *Atributos*.
- *Descripción:* ver TASA_MIN en *Atributos*.

6. SUMAR_UNO_ALARMA:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* incrementa en uno la cantidad de veces (CANT_VECES) que se evaluó la alarma.
- *Devuelve:* si la operación termina bien devuelve 0, si termina mal devuelve -1.
- *Parámetros*

almid:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la posición dentro del arreglo de la condición de alarma a incrementar.

7. CALCULAR_MIN_MAX:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* se calculan los valores AL_MIN.y AL_MAX en función de un valor de set point y un porcentaje de tolerancia provistos como parámetro. Los cálculos son los siguientes:

AL_MIN = set_point-set_point*porc_tolerancia/100
AL_MAX = set_point+set_point*porc_tolerancia/100

- *Devuelve:* si la operación termina bien devuelve 0, si termina mal devuelve -1.
- *Parámetros*

almid:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la posición dentro del arreglo de la condición de alarma a calcular.

set_point:
- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el valor de set point usado para el cálculo.

porc_tolerancia:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* 0..100
- *Descripción:* representa el porcentaje de tolerancia con respecto al valor de set point usado para el cálculo.

8. BUSCAR_ALARMA:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* recorre el arreglo de relaciones secuencialmente hasta encontrar un elemento cuyos valores de *puntero a tableaux* e *índice* coincidan con los valores pasados como parámetro al método.
- *Devuelve:* el almid de la alarma relacionada a un punto de la imagen o -1 en el caso de no encontrar ninguna alarma relacionada al punto de la imagen pasado como parámetro.

- *Parámetros*
  tabla:
  - *E/S:* entrada
  - *Tipo de Datos:* puntero a TABLEAUX
  - *Rango:* ver TABLA en *Atributos.*
  - *Descripción:* ver TABLA en *Atributos.*

  índice:
  - *E/S:* entrada
  - *Tipo de Datos:* entero
  - *Rango:* ver INDICE en *Atributos.*
  - *Descripción:* ver INDICE en *Atributos.*

9. BUSCAR_PUNTO:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* recorre el arreglo de relaciones secuencialmente hasta encontrar un elemento cuyo valor de *almid* coincida con el valor pasado como parámetro al método.
- *Devuelve:* el índice de la tabla de relaciones o -1 en el caso de no encontrar ningún punto de la imagen relacionado a la alarma pasada como parámetro.
- *Parámetros*

  almid:
  - *E/S:* entrada
  - *Tipo de Datos:* entero
  - *Rango:* cualquier valor representable por el tipo de datos
  - *Descripción:* representa la posición dentro del arreglo de la condición de alarma a calcular.

  tabla:
  - *E/S:* salida
  - *Tipo de Datos:* puntero a TABLEAUX
  - *Rango:* ver TABLA en *Atributos.*
  - *Descripción:* ver TABLA en *Atributos.*

  índice:
  - *E/S:* salida
  - *Tipo de Datos:* entero
  - *Rango:* ver INDICE en *Atributos.*
  - *Descripción:* ver INDICE en *Atributos.*

10. RELACIONAR_ALARMA_IMAGEN:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* si hay espacio, agrega un registro de relación entre un punto de la imagen y una condición de alarma.
- *Devuelve:* el índice del registro agregado si la operación terminó con éxito o -1 si el *almid* pasado como parámetro es erróneo.
- *Parámetros*

  almid:
  - *E/S:* entrada
  - *Tipo de Datos:* entero
  - *Rango:* ver ALMID en *Atributos.*
  - *Descripción:* ver ALMID en *Atributos.*

  tabla:
  - *E/S:* entrada
  - *Tipo de Datos:* puntero a TABLEAUX
  - *Rango:* ver TABLA en *Atributos.*
  - *Descripción:* ver TABLA en *Atributos.*

  índice:

- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* ver INDICE en *Atributos.*
- *Descripción:* ver INDICE en *Atributos.*

11. BORRAR_ALARMA:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* si el *almid* pasado como parámetro es válido y no tiene registros de relación con ningún punto de la imagen, entonces borra la condición de alarma del arreglo.
- *Devuelve:*

0 si la operación terminó bien,
-1 si no pudo borrar en la tabla de alarmas,
1 si esta alarma tiene algún registro de relación (en este caso no borra)
- *Parámetros*

almid:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* ver ALMID en *Atributos.*
- *Descripción:* ver ALMID en *Atributos.*

12. BORRAR_RELAC:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* si el *relid* pasado como parámetro es válido, borra la relación del arreglo.
- *Devuelve:* 0 si la operación terminó bien, o -1 si la operación terminó mal.
- *Parámetros*

relid:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Descripción:* representa la posición de un registro de relación alarma-imagen dentro del arreglo.

13. LEER_RELAC:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* si el *relid* pasado como parámetro es válido, devuelve los valores del registro de relación.
- *Devuelve:* 0 si la operación terminó bien, o -1 si la operación terminó mal.
- *Parámetros*

relid:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Descripción:* representa la posición de un registro de relación alarma-imagen dentro del arreglo.

almid:
- *E/S:* salida
- *Tipo de Datos:* entero
- *Rango:* ver ALMID en *Atributos.*
- *Descripción:* ver ALMID en *Atributos.*

tabla:
- *E/S:* salida
- *Tipo de Datos:* puntero a TABLEAUX
- *Rango:* ver TABLA en *Atributos.*

- *Descripción:* ver TABLA en *Atributos.*

índice:
- *E/S:* salida
- *Tipo de Datos:* entero
- *Rango:* ver INDICE en *Atributos.*
- *Descripción:* ver INDICE en *Atributos.*

14. ARRANCAR:
- *Tipo de Acceso:* público
- *Parte de:* ALARMA
- *Descripción:* este método es el *Analizador de Alarmas* del *Motor de Ejecución.* Es lanzado en un thread independiente de ejecución por el método *LANZAR_ALARMA.* Ejecuta en un loop indefinidamente evaluando en cada ciclo el atributo booleano *CONTINUAR* para determinar si debe seguir ejecutando o debe finalizar la ejecución. Las tareas llevadas a cabo dentro del ciclo son las siguientes:

1. Bloquearse a la espera de una señal de semáforo de evento
2. Evaluar el tiempo transcurrido desde el último ciclo
3. Para cada una de las condiciones de alarma
3.1 Leer los datos de la condición de alarma
3.2 Si no hay que evaluar la condición de alarma (*tpo_transcurrido + tpo_esperado < frecuencia*) entonces actualizar el tiempo esperado.
3.3 Sino, buscar con que puntos de la imagen está relacionada la condición de alarma
3.3.1 Para cada uno de las relaciones del arreglo de relación alarma-imagen
3.3.1.1 Si la relación involucra a la condición de alarma en evaluación
3.3.1.1.1 Leer los valores del punto de la imagen seleccionado
3.3.1.1.2 Si el punto de la imagen relacionado es digital y la cantidad de referencias es distinta de cero entonces:
3.3.1.1.2.1 Inicializar la cantidad de referencias y el estado del punto de la imagen
3.3.1.1.2.2 Inicializar el registro de parámetros y lanzar un thread independiente con la rutina de atención de interrupción que corresponda
3.3.1.1.3 Si el punto no es digital, se debe chequear el estado de alarma en función del criterio de evaluación:
3.3.1.1.3.1 Si el criterio es alta y el valor supera al_max (*valor > al_max*)
3.3.1.1.3.1.1 Inicializar la cantidad de referencias y el estado del punto de la imagen
3.3.1.1.3.1.2 Inicializar el registro de parámetros y lanzar un thread independiente con la rutina de atención de interrupción que corresponda
3.3.1.1.3.2 Si el criterio es baja y el valor es menor que al_min (*valor < al_min*)
3.3.1.1.3.2.1 Inicializar la cantidad de referencias y el estado del punto de la imagen
3.3.1.1.3.2.2 Inicializar el registro de parámetros y lanzar un thread independiente con la rutina de atención de interrupción que corresponda
3.3.1.1.3.3 Si el criterio es tasa y la cantidad de referencias es menor que la tasa mínima o mayor que la tasa máxima (*cant_ref < tasa_min*) || (*cant_ref > tasa_max*)
3.3.1.1.3.3.1 Inicializar la cantidad de referencias y el estado del punto de la imagen
3.3.1.1.3.3.2 Inicializar el registro de parámetros y lanzar un thread independiente con la rutina de atención de interrupción que corresponda
3.3.1.1.4 Incrementar en 1 la cantidad de veces que se evaluó la alarma e inicializar el tiempo esperado de la alarma.

- *Devuelve:* no devuelve ningún valor.
- *Parámetros:* no tiene.

15. LANZAR_ALARMA:
- *Tipo de Acceso:* público virtual
- *Parte de:* ALARMA
- *Descripción:* este método esta relacionada estrictamente con cuestiones de implementación del *Analizador de Alarmas.* Es por este motivo que fue definido como virtual, para permitir una

redefinición en caso de que fuera necesario por cuestiones de implementación (por ej.: este método crea un thread para lanzar el método ARRANCAR; los threads son una implementación del OS/2, si se quisiera migrar a otro sistema operativo, el trabajo se minimizaría). Los recursos usados por IGNATIUS que están ligados a la implementación de este caso en particular fueron, en su gran mayoría, identificados y encapsulados dentro de métodos virtuales. Las tareas llevadas a cabo por este método son las siguientes:
1. Inicialización de variables
2. Creación de un semáforo de eventos para el *Planificador de Tareas*.
3. Creación de un thread para la ejecución del método ARRANCAR
Devuelve: un puntero a un objeto de tipo IThread (UICL). Este objeto es el thread en el cual ejecuta el método ARRANCAR de la clase.

- *Parámetros*
  semáforo:
  - *E/S:* salida
  - *Tipo de Datos:* semáforo de eventos (HEV)
  - *Rango:* cualquier valor representable por el tipo de datos.
  - *Descripción:* representa el valor del manejador del semáforo de eventos usado por la clase TAREA para planificar la ejecución del *Analizador de Alarmas*.
  p_ventana:
  - *E/S:* entrada
  - *Tipo de Datos:* puntero a IWindow (UICL)
  - *Rango:* cualquier valor representable por el tipo de datos.
  - *Descripción:* este parámetro es usado para pasar a las rutinas de atención de usuario un puntero a una ventana de OS/2. Esto es útil en el caso de que alguna rutina de atención de usuario necesite crear alguna ventana PM, ya que para ello, necesita definir cuál es la ventana *owner* [ver OS/2 Presentation Manager Guide and Reference].

16 DETENER_ALARMA
- *Tipo de Acceso:* público virtual
- *Parte de:* ALARMA
- *Descripción:* pone en falso el atributo booleano *CONTINUAR* para detener la ejecución del *Analizador de Alarmas*.
- *Devuelve:* no devuelve ningún valor.
- *Parámetros: no tiene.*

17 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* ALARMA
- *Descripción:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Devuelve:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

18 SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* ALARMA
- *Descripción:* ver *SALIRCRITSEC* en *tableaux_int*
- *Devuelve:* ver *SALIRCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.6 CLASE cola

La clase *cola* representa el mecanismo de comunicación entre la imagen y el mundo exterior. Almacena los valores que van desde el mundo exterior hacia la imagen o viceversa. La implementación es usando listas encadenadas con política FIFO (*first-in first-out*).

La siguiente es la definición de la clase *cola* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class cola {
  struct reg_cola {
    tableaux* tabla;
    int indice;
    double valor;
    short estado;
    char timestamp[20];
    reg_cola* proximo;
  };
  reg_cola* primero;
  reg_cola* ultimo;
  reg_cola* auxiliar;
public:
  cola();
  ~cola();
  virtual void agregar_elemento(tableaux* tabla,int indice,double valor,short estado);
  virtual int sacar_elemento(tableaux*& tabla,int& indice,double& valor,short& estado);
  virtual void listar_cola();
protected:
  virtual unsigned long EntrarCritSec();
  virtual unsigned long SalirCritSec();
};
```

La clase *cola* posee los siguientes **atributos**:

1. REG_COLA:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* COLA
- *Descripción:* representa la estructura de un elemento de la cola.

1.1 TABLA:
- *Tipo de Datos:* puntero a TABLEAUX
- *Tipo de Acceso:* privado
- *Rango:* ver *TABLA* en *alarma*
- *Parte de:* REG_COLA
- *Descripción:* ver *TABLA* en *alarma*

1.2 INDICE:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* ver *INDICE* en *alarma*
- *Parte de:* REG_COLA
- *Descripción:* ver *INDICE* en *alarma*

1.3 VALOR:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* ver *VALOR* en *tableaux_int*
- *Parte de:* REG_COLA
- *Descripción:* ver *VALOR* en *tableaux_int*

1.4 ESTADO:
- *Tipo de Datos:* short
- *Tipo de Acceso:* privado
- *Rango* ver *ESTADO* en *tableaux_int*
- *Parte de:* REG_COLA
- *Descripción:* ver *ESTADO* en *tableaux_int*

1.5 TIMESTAMP:
- *Tipo de Datos:* caracter de 20 posiciones
- *Tipo de Acceso:* privado
- *Rango:* ver *TIMESTAMP* en *tableaux_int*
- *Parte de:* REG_COLA
- *Descripción:* ver *TIMESTAMP* en *tableaux_int*

1.6 PROXIMO:
- *Tipo de Datos:* puntero a REG_COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* REG_COLA
- *Descripción:* almacena un puntero al próximo elemento en la cola.

2 PRIMERO:
- *Tipo de Datos:* puntero a REG_COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* COLA
- *Descripción:* almacena un puntero al primer elemento de la cola.

3 ULTIMO:
- *Tipo de Datos:* puntero a REG_COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* COLA
- *Descripción:* almacena un puntero al último elemento de la cola.

4 AUXILIAR:
- *Tipo de Datos:* puntero a REG_COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* COLA
- *Descripción:* almacena un puntero auxiliar a un elemento de la cola.

La clase *cola* posee los siguientes **métodos**:

1. COLA (constructor):
- *Tipo de Acceso:* público
- *Parte de:* COLA
- *Descripción:* es el constructor de la clase. Inicializa los valores de *PRIMERO* y *ULTIMO* para que apunten a NULL.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*: no tiene.

2. ~COLA (destructor):
- *Tipo de Acceso:* público
- *Parte de:* COLA

- *Descripción:* es el destructor de la clase. Libera el espacio alocado dinámicamente.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_ELEMENTO:
- *Tipo de Acceso:* público virtual
- *Parte de:* COLA
- *Descripción:* crea un elemento y lo encadena a la lista en la última posición.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*

tabla:
- *E/S:* entrada
- *Tipo de Datos:* puntero a TABLEAUX
- *Rango:* ver *TABLA* en *alarma*
- *Descripción:* ver *TABLA* en *alarma*

índice:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* ver *INDICE* en *alarma*
- *Descripción:* ver *INDICE* en *alarma*

valor:
- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* ver *VALOR* en *tableaux_int*
- *Descripción:* ver *VALOR* en *tableaux_int*

estado:
- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* ver *ESTADO* en *tableaux_int*
- *Descripción:* ver *ESTADO* en *tableaux_int*

4. SACAR_ELEMENTO:
- *Tipo de Acceso:* público virtual
- *Parte de:* COLA
- *Descripción:* devuelve en los argumentos los valores del primer elemento de la cola y luego lo deletea liberando el espacio por él alocado..
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*

tabla:
- *E/S:* salida
- *Tipo de Datos:* puntero a TABLEAUX
- *Rango:* ver *TABLA* en *alarma*
- *Descripción:* ver *TABLA* en *alarma*

índice:
- *E/S:* salida
- *Tipo de Datos:* entero
- *Rango:* ver *INDICE* en *alarma*
- *Descripción:* ver *INDICE* en *alarma*

valor:
- *E/S:* salida
- *Tipo de Datos:* double
- *Rango:* ver *VALOR* en *tableaux_int*
- *Descripción:* ver *VALOR* en *tableaux_int*

estado:
- *E/S:* salida

- *Tipo de Datos:* short
- *Rango:* ver *ESTADO* en *tableaux_int*
- *Descripción:* ver *ESTADO* en *tableaux_int*

5 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* COLA
- *Descripción:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Devuelve:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

6 SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* COLA
- *Descripción:* ver *SALIRCRITSEC* en *tableaux_int*
- *Devuelve:* ver *SALIRCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.7 CLASE histórico

La clase *histórico* es la encargada de almacenar, basándose en la cola de entrada, los registros históricos de actualización de los valores de la imagen en función del porcentaje de actualización especificado por el operador en tiempo de ejecución [ver PORC_ACT en *tableaux_int*]. Estos registros son almacenados en un archivo en disco.
La siguiente es la definición de la clase *histórico* que se encuentra dentro del archivo header de la biblioteca [ver USANDO LA LIBRERÏA]:

```
class historico {
   FILE* archivo_historico;
   char* sarchivo;
   cola* cola_a_procesar;
   Boolean continuar;
public:
   historico(char* s="historic.dat");
   ~historico();
   int agregar_registro(tableaux* tabla,int indice,double valor,short estado);
   long comenzar_lectura();
   int leer_historico(char* linea);
   void finalizar_lectura(long pos_inicial);
   int listar_historico(char* dest="historic.prn");
   int calcular_tendencia(tableaux* tabla,int indice);
   void procesar_cola();
   virtual IThread* lanzar_historico(cola* c1,HEV& semaforo);
   virtual void detener_historico();
};
```

La clase *histórico* posee los siguientes **atributos**:

1 ARCHIVO_HISTORICO:
- *Tipo de Datos:* puntero a FILE
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* HISTORICO
- *Descripción:* es un puntero al archivo de datos históricos usado por las funciones de I/O de archivos.

2 SARCHIVO:
- *Tipo de Datos:* puntero a caracter
- *Tipo de Acceso:* privado

- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* HISTORICO
- *Descripción:* es un puntero a la cadena de caracteres que contiene el nombre físico del archivo de datos históricos.

3 COLA_A_PROCESAR:
- *Tipo de Datos:* puntero a objeto COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* HISTORICO
- *Descripción:* es un puntero al objeto que representa la cola de entrada. Esta cola es procesada por el *Actualizador de Histórico* del motor de ejecución.

4. CONTINUAR:
- *Tipo de Datos:* boolean
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* HISTORICO
- *Descripción:* es el atributo evaluado por el motor de ejecución de HISTORICO para determinar si debe continuar o debe finalizar su ejecución.

La clase *histórico* posee los siguientes **métodos**:

1. HISTORICO (constructor):
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO
- *Descripción:* es el constructor de la clase. Inicializa atributos y abre el archivo en modo append.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*:

  s:
  - *E/S:* entrada
  - *Tipo de Datos:* puntero a caracter
  - *Rango:* cualquier valor representable por el tipo de datos.
  - *Descripción:* es un puntero a la cadena de caracteres que contiene el nombre físico del archivo de datos históricos. Si no se provee ningún valor en el momento en que se invoca el método, el valor de este parámetro se instancia con un puntero a la cadena "historic.dat".

2. ~HISTORICO (destructor):
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO
- *Descripción:* es el destructor de la clase. Cierra el archivo abierto por el constructor de la clase.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_REGISTRO:
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO
- *Descripción:* compone y agrega un registro de actualización de valores de la imagen.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*

  tabla:
  - *E/S:* entrada
  - *Tipo de Datos:* puntero a TABLEAUX
  - *Rango:* ver *TABLA* en *alarma*
  - *Descripción:* ver *TABLA* en *alarma*

índice:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* ver *INDICE* en *alarma*
- *Descripción:* ver *INDICE* en *alarma*

valor:
- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* ver *VALOR* en *tableaux_int*
- *Descripción:* ver *VALOR* en *tableaux_int*

estado:
- *E/S:* entrada
- *Tipo de Datos:* short
- *Rango:* ver *ESTADO* en *tableaux_int*
- *Descripción:* ver *ESTADO* en *tableaux_int*

4. COMENZAR_LECTURA:
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO
- *Descripción:* prepara el archivo, cursor, para comenzar la lectura del mismo.
- *Devuelve:* la posición del cursor al momento de comenzar la lectura.
- *Parámetros: no tiene.*

5. LEER_HISTORICO:
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO
- *Descripción:* lee un registro del archivo histórico y lo devuelve en el parámetro *línea.*
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*

línea:
- *E/S:* salida
- *Tipo de Datos:* puntero a caracter
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* almacena un puntero a una cadena de caracteres que contiene el registro leído del archivo histórico.

6. FINALIZAR_LECTURA:
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO
- *Descripción:* posiciona el cursor en la posición especificada en el parámetro *pos_inicial*. Este método debe ser usado una vez que se haya terminado de ejecutar el método *LEER_HISTORICO,* especificando en el parámetro *pos_inicial* el valor devuelto por el método *COMENZAR_LECTURA*. De esta manera se restablece la posición del cursor a la que tenía antes de comenzar la lectura.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*

pos_inicial:
- *E/S:* entrada
- *Tipo de Datos:* long
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* almacena el valor donde se desea posicionar el cursor.

7. LISTAR_HISTORICO:
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO

- *Descripción:* copia el contenido del histórico en el archivo especificado como parámetro. Debido a que en OS/2 los ports pueden ser tratados como archivos, si se especifica como parámetro el nombre de un port conectado a una impresora, el resultado será un listado por impresora del archivo histórico.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*

dest:
- *E/S:* entrada
- *Tipo de Datos:* puntero a caracter
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* almacena un puntero a una cadena de caracteres que contiene el nombre del archivo en el cual se quiere copiar el contenido del archivo histórico. Si no se especifica ningún valor, el contenido será copiado en un archivo cuyo nombre es "historic.prn".

8. CALCULAR_TENDENCIA:
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO
- *Descripción:* dado un punto de la imagen, devuelve todos los valores históricos asociados a ese punto. Este método esta en construcción.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*

tabla:
- *E/S:* entrada
- *Tipo de Datos:* puntero a TABLEAUX
- *Rango:* ver *TABLA* en *alarma*
- *Descripción:* ver *TABLA* en *alarma*

índice:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* ver *INDICE* en *alarma*
- *Descripción:* ver *INDICE* en *alarma*

9. PROCESAR_COLA:
- *Tipo de Acceso:* público
- *Parte de:* HISTORICO
- *Descripción:* este método es el *Actualizador de Histórico* del motor de ejecución. Es lanzado en un thread independiente de ejecución por el método *LANZAR_HISTORICO*. Ejecuta en un loop indefinidamente evaluando en cada ciclo el atributo booleano *CONTINUAR* para determinar si debe seguir ejecutando o debe finalizar la ejecución. Las tareas llevadas a cabo dentro del ciclo son las siguientes:

1. Bloquearse a la espera de una señal de semáforo de evento
2. Mientras se pueda sacar un elemento de la cola
2.1 Sacar el primer elemento de la cola.
2.2 Leer todos los atributos del punto de la imagen relacionado a ese elemento.
2.3 Modificar en la imagen el *VALOR* y el *ESTADO* del punto.
2.4 Si es necesario actualizar el histórico:
Para ello se debe verificar:
a. el porcentaje de actualización definido para ese punto de la imagen
b. si cambio el estado
Si ((valor > (v1 * (100 + porc_act) / 100)) ||
(valor < (v1 * (100 - porc_act) / 100)) || (estado != e1))
2.4.1 Agregar registro en el archivo histórico con los nuevos valores
2.5 Si cambió el valor del punto, entonces:
2.5.1 Sumar 1 a la cantidad de referencias hechas a ese punto de la imagen
- *Devuelve:* no devuelve ningún valor.
- *Parámetros:* no tiene.

10. LANZAR_HISTORICO:
- *Tipo de Acceso:* público virtual
- *Parte de:* HISTORICO
- *Descripción:* este método está relacionado estrictamente con cuestiones de implementación del *Actualizador de Histórico*. Es por este motivo que fue definido como virtual, para permitir una redefinición en caso de que fuera necesario por cuestiones de implementación (por ej.: este método crea un thread para lanzar el método PROCESAR_COLA; los threads son una implementación del OS/2, si se quisiera migrar a otro sistema operativo, el trabajo se minimizaría). Los recursos usados por IGNATIUS que están ligados a la implementación de este caso en particular fueron, en su gran mayoría, identificados y encapsulados dentro de métodos virtuales. Las tareas llevadas a cabo por este método son las siguientes:
1. Inicialización de variables
2. Creación de un semáforo de eventos para el *Actualizador de Histórico*.
3. Creación de un thread para la ejecución del método PROCESAR_COLA
- *Devuelve:* un puntero a un objeto de tipo IThread (UICL). Este objeto es el thread en el cual ejecuta el método PROCESAR_COLA de la clase.
- *Parámetros*

  c1:
  - *E/S:* entrada
  - *Tipo de Datos:* puntero a objeto *COLA*
  - *Rango:* cualquier valor representable por el tipo de datos.
  - *Descripción:* es un puntero a un objeto de tipo cola. Este objeto es procesado por el *Actualizador de Histórico*.

  semáforo:
  - *E/S:* salida
  - *Tipo de Datos:* semáforo de eventos (HEV)
  - *Rango:* cualquier valor representable por el tipo de datos.
  - *Descripción:* representa el valor del manejador del semáforo de eventos usado por la clase TAREA para planificar la ejecución del *Actualizador de Histórico*.

11. DETENER_HISTORICO
- *Tipo de Acceso:* público virtual
- *Parte de:* HISTORICO
- *Descripción:* pone en falso el atributo booleano *CONTINUAR* para detener la ejecución del *Actualizador de Histórico*.
- *Devuelve:* no devuelve ningún valor.
- *Parámetros: no tiene.*

2.4.2.8 CLASE tarea

La clase *tarea* es la encargada de planificar la ejecución de los distintos componentes del motor de ejecución. Su uso puede extenderse para planificar rutinas escritas por el usuario, como por ejemplo una rutina de refresco de datos en pantalla. Ejecuta una planificación de alto nivel de todas las rutinas que ejecutan dentro del supervisor, basándose en los tiempos de período especificados para cada una de ellas.

La siguiente es la definición de la clase *tarea* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class tarea {
  struct reg_tarea {
    char nombre[12];
    int proc_ind;
```

```
    unsigned long int tpo_periodo, tpo_esperado;
    HEV sem_handle;
    char timestamp[20];
  };
  reg_tarea* v;
  Boolean continuar;
public:
  int cant_tareas, long_tabla;
  tarea(int s=10);
  ~tarea();
  int agregar_tarea(char* nombre,int proc_ind,unsigned long int tpo_periodo,unsigned long int tpo_esperado,HEV sem_handle);
  int modificar_tarea(int indice, char* nombre,int proc_ind,unsigned long int tpo_periodo,
                unsigned long int tpo_esperado,HEV sem_handle);
  int leer_tarea(int indice,char* nombre,int& proc_ind,unsigned long int& tpo_periodo,
            unsigned long int& tpo_esperado,HEV& sem_handle);
  void mostrar_tarea();
  int borrar_tarea(int indice);
  void planificar();
  virtual IThread* arrancar_planificador();
  virtual void    detener_planificador();
protected:
  virtual unsigned long EntrarCritSec();
  virtual unsigned long SalirCritSec();
};
```

La clase *tarea* posee los siguientes **atributos**:

1. REG_TAREA:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* TAREA
- *Descripción:* representa la estructura de un elemento de la clase tarea.

1.1 NOMBRE:
- *Tipo de Datos:* cadena de caracteres de 12 posiciones
- *Tipo de Acceso:* privado
- *Rango:* cualquier cadena de caracteres de 11 posiciones que termine con el caracter de fin de cadena.
- *Parte de:* REG_TAREA
- *Descripción:* almacena el nombre de la tarea.

1.2 PROC_IND:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* REG_TAREA
- *Descripción:* representa el índice del proceso que está relacionado con la tarea

1.3 TPO_PERIODO:
- *Tipo de Datos:* unsigned long int
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* REG_TAREA
- *Descripción:* representa la frecuencia de tiempo con la que se debe enviar un mensaje de semáforo de evento a la tarea para que continúe su ejecución. Este valor está expresado en centésimas de segundo.

1.4 TPO_ESPERADO:
- *Tipo de Datos:* unsigned long int
- *Tipo de Acceso:* privado

- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* REG_TAREA
- *Descripción:* representa el tiempo transcurrido desde el último ciclo de ejecución de la tarea. Este valor está expresado en centésimas de segundo.

1.5 SEM_HANDLE:
- *Tipo de Datos:* HEV
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* REG_TAREA
- *Descripción:* representa el manejador del semáforo de evento correspondiente a la tarea.

1.6 TIMESTAMP:
- *Tipo de Datos:* caracter de 20 posiciones
- *Tipo de Acceso:* privado
- *Rango:* ver *TIMESTAMP* en *tableaux_int.*
- *Parte de:* REG_TAREA
- *Descripción:* representa la fecha y hora en la cual fue modificado por última vez algún valor del registro de tarea.

2. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_TAREA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TAREA
- *Descripción:* cuando se crea un objeto se reserva espacio para n tareas (elementos de tipo REG_TAREA). V es el puntero a este espacio de memoria.

3. CONTINUAR:
- *Tipo de Datos:* boolean
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* TAREA
- *Descripción:* es el atributo evaluado por el *Planificador de Tareas* del motor de ejecución para determinar si debe continuar o debe finalizar su ejecución.

4. CANT_TAREAS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* TAREA
- *Descripción:* representa la cantidad de tareas almacenadas en el objeto.

5. LONG_TABLA
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* TAREA
- *Descripción:* representa la cantidad máxima de tareas que pueden ser almacenadas por el objeto.

La clase *tarea* posee los siguientes **métodos**:

1. TAREA (constructor):
- *Tipo de Acceso:* público
- *Parte de:* TAREA

- *Descripción:* es el constructor de la clase. Inicializa atributos y reserva espacio para almacenar las tareas.. El espacio reservado para las tareas es alocado dinámicamente con el formato de un arreglo de *s* elementos de tipo REG_TAREA. Se puede invocar pasando como parámetro un número entero *s* para indicar la cantidad de tareas y relaciones que el objeto va a almacenar como máximo; en caso contrario, se reserva espacio para un máximo de 10 tareas.
- *Devuelve:* no devuelve ningún valor
- *Parámetros:*

  s:
  - *E/S:* entrada
  - *Tipo de Datos:* entero
  - *Rango:* cualquier valor positivo representable por el tipo de datos
  - *Descripción:* se utiliza para indicar la cantidad de tareas que debe almacenar el objeto como máximo.

2. ~TAREA (destructor):
- *Tipo de Acceso:* público
- *Parte de:* TAREA
- *Descripción:* es el destructor de la clase. Libera el espacio alocado por el constructor de la clase.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_TAREA:
- *Tipo de Acceso:* público
- *Parte de:* TAREA
- *Descripción:* si hay espacio en el arreglo, agrega un registro de tarea.
- *Devuelve:* el índice del arreglo donde agregó el elemento o -1 si no pudo realizar la operación por falta de espacio..
- *Parámetros*

  nombre:
  - *E/S:* entrada
  - *Tipo de Datos:* puntero a cadena de caracteres
  - *Rango:* ver *NOMBRE* en Atributos
  - *Descripción:* ver *NOMBRE* en Atributos

  proc_ind:
  - *E/S:* entrada
  - *Tipo de Datos:* ver *PROC_IND* en Atributos
  - *Rango:* ver *PROC_IND* en Atributos
  - *Descripción:* ver *PROC_IND* en Atributos

  tpo_periodo:
  - *E/S:* entrada
  - *Tipo de Datos:* ver *TPO_PERIODO* en Atributos
  - *Rango:* ver *TPO_PERIODO* en Atributos
  - *Descripción:* ver *TPO_PERIODO* en Atributos

  tpo_esperado:
  - *E/S:* entrada
  - *Tipo de Datos:* ver *TPO_ESPERADO* en Atributos
  - *Rango:* ver *TPO_ESPERADO* en Atributos
  - *Descripción:* ver *TPO_ESPERADO* en Atributos

  sem_handle:
  - *E/S:* entrada
  - *Tipo de Datos:* ver *SEM_HANDLE* en Atributos
  - *Rango:* ver *SEM_HANDLE* en Atributos
  - *Descripción:* ver *SEM_HANDLE* en Atributos

4. MODIFICAR_TAREA:

- *Tipo de Acceso:* público
- *Parte de:* TAREA
- *Descripción:* modifica los valores del registro de tarea especificado como parámetro en *índice*.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros:*

indice:
- *E/S:* entrada
- *Tipo de Datos: entero*
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* representa la posición del arreglo donde se encuentra el elemento que se quiere modificar.

nombre: ver *AGREGAR_TAREA*
proc_ind: ver *AGREGAR_TAREA*
tpo_periodo: ver *AGREGAR_TAREA*
tpo_esperado: ver *AGREGAR_TAREA*
sem_handle: ver *AGREGAR_TAREA*

5. LEER_TAREA:
- *Tipo de Acceso:* público
- *Parte de:* TAREA
- *Descripción:* devuelve en los parámetros de salida los valores del registro de tarea especificado como parámetro en *índice*.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*

indice:
- *E/S:* entrada
- *Tipo de Datos: entero*
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* representa la posición del arreglo donde se encuentra el elemento que se quiere leer.

nombre:
- *E/S:* salida
- *Tipo de Datos:* puntero a cadena de caracteres
- *Rango:* ver *NOMBRE* en Atributos
- *Descripción:* ver *NOMBRE* en Atributos

proc_ind:
- *E/S:* salida
- *Tipo de Datos:* ver *PROC_IND* en Atributos
- *Rango:* ver *PROC_IND* en Atributos
- *Descripción:* ver *PROC_IND* en Atributos

tpo_periodo:
- *E/S:* salida
- *Tipo de Datos:* ver *TPO_PERIODO* en Atributos
- *Rango:* ver *TPO_PERIODO* en Atributos
- *Descripción:* ver *TPO_PERIODO* en Atributos

tpo_esperado:
- *E/S:* salida
- *Tipo de Datos:* ver *TPO_ESPERADO* en Atributos
- *Rango:* ver *TPO_ESPERADO* en Atributos
- *Descripción:* ver *TPO_ESPERADO* en Atributos

sem_handle:
- *E/S:* salida
- *Tipo de Datos:* ver *SEM_HANDLE* en Atributos
- *Rango:* ver *SEM_HANDLE* en Atributos

- *Descripción:* ver *SEM_HANDLE* en Atributos

6. BORRAR_TAREA:
- *Tipo de Acceso:* público
- *Parte de:* TAREA
- *Descripción:* elimina del arreglo el registro de tarea especificado como parámetro.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*

indice:
- *E/S:* entrada
- *Tipo de Datos: entero*
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* representa la posición del arreglo donde se encuentra el elemento que se quiere borrar.

7. PLANIFICAR:
- *Tipo de Acceso:* público
- *Parte de:* TAREA
- *Descripción:* este método es el *Planificador de Tareas* del motor de ejecución. Es lanzado en un thread independiente de ejecución por el método *ARRANCAR_PLANIFICADOR*. Ejecuta en un loop indefinidamente evaluando en cada ciclo el atributo booleano *CONTINUAR* para determinar si debe seguir ejecutando o debe finalizar la ejecución. Las tareas llevadas a cabo dentro del ciclo son las siguientes:

1. Evaluar el tiempo transcurrido desde el último ciclo.
2. Para cada una de las tareas
2.1 Leer los valores de los atributos del registro de tarea.
2.2 Si el tiempo transcurrido más el tiempo esperado por la tarea es igual o mayor que el tiempo de período de la tarea, entonces *((tpo_transcurrido + tpo_esperado) >= tpo_periodo)*:
2.2.1 Inicializar el tiempo esperado de la tarea *(tpo_esperado = 0)*
2.2.2 Despertar a la tarea a través del semáforo
2.3 Sino:
2.3.1 Incrementar el Tiempo Esperado de la tarea *(tpo_esperado += tpo_transcurrido)*

Cuando sale del loop hace envía un mensaje de semáforo de evento a todas y c/u de las tareas para asegurar que no queda ninguna bloqueada a la espera de un mensaje de semáforo de evento.
- *Devuelve:* no devuelve ningún valor.
- *Parámetros:* no tiene.

8. ARRANCAR_PLANIFICADOR:
- *Tipo de Acceso:* público virtual
- *Parte de:* TAREA
- *Descripción:* este método está relacionado estrictamente con cuestiones de implementación del *Planificador de Tareas*. Es por este motivo que fue definido como virtual, para permitir una redefinición en caso de que fuera necesario por cuestiones de implementación (por ej.: este método crea un thread para lanzar el método PLANIFICAR; los threads son una implementación del OS/2, si se quisiera migrar a otro sistema operativo, el trabajo se minimizaría). Los recursos usados por IGNATIUS que están ligados a la implementación de este caso en particular fueron, en su gran mayoría, identificados y encapsulados dentro de métodos virtuales. Las tareas llevadas a cabo por este método son las siguientes:

1. Inicialización de variables
2. Creación de un thread para la ejecución del método PLANIFICAR

- *Devuelve:* un puntero a un objeto de tipo IThread (UICL). Este objeto es el thread en el cual ejecuta el método PLANIFICAR de la clase.
- *Parámetros:* no tiene

9 DETENER_PLANIFICADOR:

- *Tipo de Acceso:* público virtual
- *Parte de:* TAREA
- *Descripción:* pone en falso el atributo booleano *CONTINUAR* para detener la ejecución del *Planificador de Tareas*.
- *Devuelve:* no devuelve ningún valor.
- *Parámetros: no tiene.*

5 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* TAREA
- *Descripción:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Devuelve:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

6 SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* TAREA
- *Descripción:* ver *SALIRCRITSEC* en *tableaux_int*
- *Devuelve:* ver *SALIRCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.9 CLASE proceso

La clase *proceso* es la encargada de modelar los procesos existentes en el mundo real, como por ej.: proceso de secado de materia prima textil, proceso de llenado y vaciado de fluidos, etc..

La siguiente es la definición de la clase *proceso* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class proceso {
  struct reg_proceso {
    char nombre[12];
    int proc_id, bitmap_id;
    unsigned long int frec_disp, tpo_esp;
    char descrip[40];
    char timestamp[20];
  };
  reg_proceso* v;
public:
  int cant_procesos, long_tabla;
  int proceso_en_display;
  proceso(int s=10);
  ~proceso();
  int agregar_proceso(int proc_id,char* nombre,int bitmap_id,unsigned long int frec_disp,char* descrip,unsigned long int tpo_esp);
  int modificar_proceso(int proc_ind,int proc_id,char* nombre,int bitmap_id,unsigned long int frec_disp,char* descrip,unsigned long
                        int tpo_esp);
  int leer_proceso(int proc_ind,int& proc_id,char* nombre,int& bitmap_id,unsigned long int& frec_disp,char *descrip,unsigned long
int&
                   tpo_esp);
  void mostrar_proceso();
  int borrar_proceso(int proc_ind);
  int modificar_frecuencia(int proc_ind,unsigned long int frec_disp);
  int existe_proceso(int proc_id);
protected:
  virtual unsigned long EntrarCritSec();
  virtual unsigned long SalirCritSec();
};
```

La clase *proceso* posee los siguientes **atributos**:

1. REG_PROCESO:

- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* PROCESO
- *Descripción:* representa la estructura de un elemento de la clase proceso.

1.1 NOMBRE:
- *Tipo de Datos:* cadena de caracteres de 12 posiciones
- *Tipo de Acceso:* privado
- *Rango:* cualquier cadena de caracteres de 11 posiciones que termine con el caracter de fin de cadena.
- *Parte de:* REG_PROCESO
- *Descripción:* almacena el nombre del proceso.

1.2 PROC_ID:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* REG_PROCESO
- *Descripción:* representa una identificación numérica del proceso.

1.3 BITMAP_ID:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* REG_PROCESO
- *Descripción:* representa el resource-id del bitmap asociado al proceso.

1.4 FREC_DISP:
- *Tipo de Datos:* unsigned long int
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* REG_PROCESO
- *Descripción:* representa la frecuencia de tiempo con la que deben refrescarse en la pantalla. Los valores de los puntos de la imagen relacionados con el proceso. Este valor está expresado en centésimas de segundo.

1.5 TPO_ESP:
- *Tipo de Datos:* unsigned long int
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* REG_PROCESO
- *Descripción:* representa el tiempo transcurrido desde la última vez que se refrescaron en pantalla los valores de los puntos de la imagen relacionados con el proceso. Este valor está expresado en centésimas de segundo.

1.6 DESCRIP:
- *Tipo de Datos:* caracter de 40 posiciones
- *Tipo de Acceso:* privado
- *Rango:* cualquier secuencia de 40 caracteres que finalice con el caracter de fin de cadena.
- *Parte de:* REG_PROCESO
- *Descripción:* almacena una descripción del proceso.

1.7 TIMESTAMP:
- *Tipo de Datos:* caracter de 20 posiciones
- *Tipo de Acceso:* privado
- *Rango:* ver *TIMESTAMP* en *tableaux_int.*
- *Parte de:* REG_PROCESO
- *Descripción:* representa la fecha y hora en la cual fue modificado por última vez algún valor del registro de proceso.

2. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_PROCESO
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* PROCESO
- *Descripción:* cuando se crea un objeto se reserva espacio para n procesos (elementos de tipo REG_PROCESO). V es el puntero a este espacio de memoria.

3. CANT_PROCESOS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* PROCESO
- *Descripción:* representa la cantidad de procesos almacenados en el objeto.

4. LONG_TABLA
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* PROCESO
- *Descripción:* representa la cantidad máxima de procesos que pueden ser almacenados por el objeto.

5. PROCESO_EN_DISPLAY
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* PROCESO
- *Descripción:* almacena el índice del arreglo correspondiente al proceso que se encuentra en display.

La clase *proceso* posee los siguientes **métodos**:

1. PROCESO (constructor):
- *Tipo de Acceso:* público
- *Parte de:* PROCESO
- *Descripción:* es el constructor de la clase. Inicializa atributos y reserva espacio para almacenar los procesos.. El espacio reservado para las procesos es alocado dinámicamente con el formato de un arreglo de *s* elementos de tipo REG_PROCESO. Se puede invocar pasando como parámetro un número entero *s* para indicar la cantidad de procesos y relaciones que el objeto va a almacenar como máximo; en caso contrario, se reserva espacio para un máximo de 10 procesos.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*:

  s:
  - *E/S:* entrada
  - *Tipo de Datos:* entero
  - *Rango:* cualquier valor positivo representable por el tipo de datos
  - *Descripción:* se utiliza para indicar la cantidad de procesos que debe almacenar el objeto como máximo.

2. ~PROCESO (destructor):
- *Tipo de Acceso:* público
- *Parte de:* PROCESO
- *Descripción:* es el destructor de la clase. Libera el espacio alocado por el constructor de la clase.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_PROCESO:
- *Tipo de Acceso:* público
- *Parte de:* PROCESO
- *Descripción:* agrega un registro en el arreglo si es que no existe ningún registro de proceso con el mismo *proc_id* del nuevo registro y si es que hay espacio para agregar un nuevo elemento.
- *Devuelve:* si puede agregar devuelve el índice del arreglo donde agregó el elemento, si NO hay más espacio para agregar devuelve - 1 y si ya existe ese *proc_id* devuelve -2
- *Parámetros*:

proc_id:
- *E/S:* entrada
- *Tipo de Datos:* ver *PROC_ID* en *Atributos*.
- *Rango:* ver *PROC_ID* en *Atributos*.
- *Descripción:* ver *PROC_ID* en *Atributos*.

nombre:
- *E/S:* entrada
- *Tipo de Datos:* puntero a cadena de caracteres.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver *NOMBRE* en *Atributos*.

bitmap_id:
- *E/S:* entrada
- *Tipo de Datos:* ver *BITMAP_ID* en *Atributos*.
- *Rango:* ver *BITMAP_ID* en *Atributos*.
- *Descripción:* ver *BITMAP_ID* en *Atributos*.

frec_disp:
- *E/S:* entrada
- *Tipo de Datos:* ver *FREC_DISP* en *Atributos*.
- *Rango:* ver *FREC_DISP* en *Atributos*.
- *Descripción:* ver *FREC_DISP* en *Atributos*.

descrip:
- *E/S:* entrada
- *Tipo de Datos:* puntero a cadena de caracteres.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver *DESCRIP* en *Atributos*.

tpo_esp:
- *E/S:* entrada
- *Tipo de Datos:* ver *TPO_ESP* en *Atributos*.
- *Rango:* ver *TPO_ESP* en *Atributos*.
- *Descripción:* ver *TPO_ESP* en *Atributos*.

4. MODIFICAR_PROCESO:
- *Tipo de Acceso:* público
- *Parte de:* PROCESO
- *Descripción:* si el índice de proceso pasado como parámetro en *proc_ind* es válido, modifica los valores del registro de proceso con los nuevos valores provistos como parámetros.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*:

proc_ind:

- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Descripción:* representa la posición dentro del arreglo del registro que se quiere modificar.

proc_id: ver *AGREGAR_PROCESO*
nombre: ver *AGREGAR_PROCESO*
bitmap_id: ver *AGREGAR_PROCESO*
frec_disp: ver *AGREGAR_PROCESO*
descrip: ver *AGREGAR_PROCESO*
tpo_esp: ver *AGREGAR_PROCESO*

5. LEER_PROCESO:
- *Tipo de Acceso:* público
- *Parte de:* PROCESO
- *Descripción:* devuelve en los parámetros los valores leídos del registro de proceso especificado en el parámetro *proc_ind.*
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*:

proc_ind:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Descripción:* representa la posición dentro del arreglo del registro que se quiere leer.

proc_id:
- *E/S:* salida
- *Tipo de Datos:* ver *PROC_ID* en *Atributos.*
- *Rango:* ver *PROC_ID* en *Atributos.*
- *Descripción:* ver *PROC_ID* en *Atributos.*

nombre:
- *E/S:* salida
- *Tipo de Datos:* puntero a cadena de caracteres.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver *NOMBRE* en *Atributos.*

bitmap_id:
- *E/S:* salida
- *Tipo de Datos:* ver *BITMAP_ID* en *Atributos.*
- *Rango:* ver *BITMAP_ID* en *Atributos.*
- *Descripción:* ver *BITMAP_ID* en *Atributos.*

frec_disp:
- *E/S:* salida
- *Tipo de Datos:* ver *FREC_DISP* en *Atributos.*
- *Rango:* ver *FREC_DISP* en *Atributos.*
- *Descripción:* ver *FREC_DISP* en *Atributos.*

descrip:
- *E/S:* salida
- *Tipo de Datos:* puntero a cadena de caracteres.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver *DESCRIP* en *Atributos.*

tpo_esp:
- *E/S:* salida
- *Tipo de Datos:* ver *TPO_ESP* en *Atributos.*
- *Rango:* ver *TPO_ESP* en *Atributos.*
- *Descripción:* ver *TPO_ESP* en *Atributos.*

6. BORRAR_PROCESO:

- *Tipo de Acceso:* público
- *Parte de:* PROCESO
- *Descripción:* si el índice de proceso pasado como parámetro es válido, borra del arreglo el registro especificado.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*:
  proc_ind: ver *LEER_PROCESO*.

7. MODIFICAR_FRECUENCIA:
- *Tipo de Acceso:* público
- *Parte de:* PROCESO
- *Descripción:* si el índice de proceso pasado como parámetro es válido, modifica la frecuencia de display del registro especificado.
- *Devuelve:* 0 si pudo realizar la operación, -1 si no pudo realizar la operación.
- *Parámetros*:
  proc_ind: ver *LEER_PROCESO*.
  frec_disp: ver *AGREGAR_PROCESO*

8. EXISTE_PROCESO:
- *Tipo de Acceso:* público
- *Parte de:* PROCESO
- *Descripción:* .recorre el arreglo y chequea si existe algún proceso con el *proc_id* especificado como parámetro.
- *Devuelve:* el *proc_ind* del proceso cuyo id es igual a *proc_id* o -1 en el caso de no encontrar ningún proceso con ese *proc_id*
- *Parámetros*:
  proc_id: ver *AGREGAR_PROCESO*

9. ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* PROCESO
- *Descripción:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Devuelve:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

10. SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* PROCESO
- *Descripción:* ver *SALIRCRITSEC* en *tableaux_int*
- *Devuelve:* ver *SALIRCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.10 CLASE MSG

La clase *MSG* es la encargada de encapsular los datos de los mensajes que envía la clase *MIMICO* al supervisor. Estos mensajes contienen información de la representación de los puntos de la imagen en la pantalla.

La siguiente es la definición de la clase *MSG* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class MSG {
public:
```

```
  unsigned long ulLongitud;
  double valor;
  int eje_x,eje_y,letra,fondo;
};
```

La clase *MSG* posee los siguientes **atributos**:

1. ULLONGITUD:
- *Tipo de Datos:* unsigned long
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* MSG
- *Descripción:* representa la longitud del objeto. Este atributo es requerido por la clase *Pipe* para determinar la cantidad de bytes que debe leer o escribir.

2 VALOR:
- *Tipo de Datos:* double
- *Tipo de Acceso:* público
- *Rango:* ver VALOR en *tableaux_int*.
- *Parte de:* MSG
- *Descripción:* ver VALOR en *tableaux_int*.

3. EJE_X:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* MSG
- *Descripción:* representa el desplazamiento relativo al eje X de la ventana donde se debe escribir el valor del punto de la imagen.

4. EJE_Y:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* MSG
- *Descripción:* representa el desplazamiento relativo al eje Y de la ventana donde se debe escribir el valor del punto de la imagen.

5. LETRA:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* MSG
- *Descripción:* representa el color de letra con el que se debe escribir el valor del punto de la imagen.

6. FONDO:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* MSG
- *Descripción:* representa el color de fondo con el que se debe escribir el valor del punto de la imagen.

La clase *MSG* no posee **métodos**:

2.4.2.11 CLASE Pipe

La clase *Pipe* es la encargada de implementar el canal de comunicaciones entre la clase *MIMICO* y el supervisor para el envío de mensajes (objetos de la clase *MSG*). Utiliza *pipes* del OS/2 para implementar este mecanismo de comunicaciones. Los *pipes* son una de las tres formas de comunicación entre procesos (IPC) que tiene el OS/2, las otras dos formas son *semáforos* y *queues* [ver OS/2 Control Program Guide and Reference].

La siguiente es la definición de la clase *Pipe* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class Pipe {
  public:
    Pipe(IString istr, IWindow * piwin);
    ULONG Create(VOID);
    ULONG Open(VOID);
    ULONG Read(MSG * pmsg);
    ULONG Write(MSG * pmsg);
    ULONG Connect(VOID);
    ULONG Disconnect(VOID);
    ULONG Close(VOID);
    ULONG Post(MPARAM mp1=0,MPARAM mp2=0);
  protected:
    VOID ErrorDisplay(IString);
  private:
    IWindow * piwin;
    IWindowHandle winh;
    IString iszPipe;
    HPIPE hpipe;
};
```

La clase *proceso* posee los siguientes **atributos**:

1. PIWIN:
- *Tipo de Datos:* puntero a IWindow (UICL)
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* PIPE
- *Descripción:* representa un puntero a un objeto de tipo IWindow. Es usado por el método *POST* para el envío de mensajes.

2. WINH:
- *Tipo de Datos:* IWindowHandle (UICL)
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* PIPE
- *Descripción:* representa el handler de un objeto de tipo IWindow. Es usado por el método *POST* para el envío de mensajes.

3. ISZPIPE:
- *Tipo de Datos:* IString (UICL)
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* PIPE
- *Descripción:* representa el nombre del pipe.

4. HPIPE:
- *Tipo de Datos:* HPIPE
- *Tipo de Acceso:* privado

- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* PIPE
- *Descripción:* representa el handler del pipe.

La clase *PIPE* posee los siguientes **métodos**:

1. PIPE (constructor):
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* es el constructor de la clase. Inicializa atributos.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*:

  istr:
  - *E/S:* entrada
  - *Tipo de Datos:* ver ISZPIPE en *Atributos*
  - *Rango:* ver ISZPIPE en *Atributos*
  - *Descripción:* ver ISZPIPE en *Atributos*

  piwin:
  - *E/S:* entrada
  - *Tipo de Datos:* ver PIWIN en *Atributos*
  - *Rango:* ver PIWIN en *Atributos*
  - *Descripción:* ver PIWIN en *Atributos*

2. CREATE:
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* crea un named pipe.
- *Devuelve:* siempre devuelve 0.
- *Parámetros*: no tiene.

3. OPEN:
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* prepara el pipe para lectura/escritura.
- *Devuelve:* siempre devuelve 0.
- *Parámetros*: no tiene.

4. READ :
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* lee un mensaje *MSG* del pipe y lo devuelve en el parámetro *pmsg*
- *Devuelve:* siempre devuelve 0.
- *Parámetros*:

  pmsg:
  - *E/S:* salida
  - *Tipo de Datos:* puntero a *MSG*
  - *Rango:* cualquier valor representable por el tipo de datos
  - *Descripción:* puntero al objeto de tipo *MSG* que contiene el mensaje leído.

5. WRITE :
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* escribe en el pipe el mensaje pasado como parámetro en *pmsg*.
- *Devuelve:* siempre devuelve 0.
- *Parámetros*:

  pmsg:

- *E/S:* entrada
- *Tipo de Datos:* puntero a *MSG*
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* puntero al objeto de tipo *MSG* que contiene el mensaje que a escribir en el pipe.

6. CONNECT :
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* conecta el pipe para poder usarlo.
- *Devuelve:* siempre devuelve 0.
- *Parámetros*: no tiene.

7. DISCONNECT :
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* desconecta el pipe para cuando se terminó de usarlo.
- *Devuelve:* siempre devuelve 0.
- *Parámetros*: no tiene.

8. CLOSE :
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* cierra el pipe cuando se terminó de usarlo.
- *Devuelve:* siempre devuelve 0.
- *Parámetros*: no tiene.

9. POST :
- *Tipo de Acceso:* público
- *Parte de:* PIPE
- *Descripción:* envía un mensaje de tipo USER con parámetros *mp1* y *mp2* a la ventana *piwin* especificada durante la creación del objeto.
- *Devuelve:* siempre devuelve 0.
- *Parámetros*:

  mp1:
  - *E/S:* entrada
  - *Tipo de Datos:* MPARAM
  - *Rango:* cualquier valor representable por el tipo de datos
  - *Descripción:* cualquier valor que el usuario quiera enviar dentro del mensaje permitido por el tipo de datos.

  mp2:
  - *E/S:* entrada
  - *Tipo de Datos:* MPARAM
  - *Rango:* cualquier valor representable por el tipo de datos
  - *Descripción:* cualquier valor que el usuario quiera enviar dentro del mensaje permitido por el tipo de datos.

10. ERRORDISPLAY :
- *Tipo de Acceso:* protegido
- *Parte de:* PIPE
- *Descripción:* es la rutina de tratamiento de errores de la clase. Muestra dentro de una ventana Presentation Manager un texto explicando el error ocurrido.
- *Devuelve:* no devuelve ningún valor.
- *Parámetros*:

  isztext:
  - *E/S:* entrada
  - *Tipo de Datos:* Istring (UICL)

- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* texto del error.

2.4.2.12 CLASE mímico

La clase *mímico* relaciona los puntos de la imagen con los distintos procesos del mundo exterior y sus valores de representación: posición en la pantalla, color de letra, color de fondo, etc. Está compuesta, además, por dos clases de implementación previamente presentadas: PIPE y MSG. Estas últimas son usadas para establecer la comunicación entre el *Administrador de Mímicos* del motor de ejecución y el supervisor.

La siguiente es la definición de la clase *mímico* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class mimico {
  struct reg_mimico {
    tableaux* tabla;
    int indice, proc_ind;
    int eje_x, eje_y, letra, fondo, letra_a, fondo_a;
  };
  reg_mimico* v;
  proceso* proc;
  HWND vent;
  IString iszPipeName;
  IWindow* paVentana;
  Pipe* paPipe;
  MSG*  paMsg;
  Boolean continuar;
public:
  int cant_mimicos, long_tabla;
  mimico(int s=10);
  ~mimico();
  int agregar_mimico(tableaux* tabla,int indice,int proc_ind,int eje_x,int eje_y,int letra,int fondo,int letra_a,int fondo_a);
  int borrar_mimico(int mim_ind);
  int leer_mimico(int mim_ind,tableaux*& tabla,int& indice,int& proc_ind,int& eje_x,int& eje_y,int& letra,int& fondo,int&
letra_a,int& fondo_a);
  int buscar_mimico(tableaux* tabla,int indice,int proc_ind);
  void mostrar_mimico();
  virtual void display();
  virtual IThread* lanzar_mimico(IWindow* ventana,proceso* proceso,IEV& semaforo,IString pipename);
  virtual void detener_mimico();
protected:
  virtual void mostrar_punto(double valor,int eje_x,int eje_y,int letra,int fondo);
  virtual unsigned long EntrarCritSec();
  virtual unsigned long SalirCritSec();
};
```

La clase *proceso* posee los siguientes **atributos:**

1. REG_MIMICO:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* MIMICO
- *Descripción:* representa la estructura de representación por pantalla de un punto de la imagen.

1.1 TABLA:
- *Tipo de Datos:* ver TABLA en *Alarma.*
- *Tipo de Acceso:* privado
- *Rango:* ver TABLA en *Alarma.*
- *Parte de:* REG_MIMICO

- *Descripción:* ver TABLA en *Alarma*.

1.2 INDICE:
- *Tipo de Datos:* ver INDICE en *Alarma*.
- *Tipo de Acceso:* privado
- *Rango:* ver INDICE en *Alarma*.
- *Parte de:* REG_MIMICO
- *Descripción:* ver INDICE en *Alarma*.

1.3 PROC_IND:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:*cualquier valor positivo representable por el tipo de datos
- *Parte de:* REG_MIMICO
- *Descripción:* índice al proceso relacionado al punto de la imagen.

1.4 EJE_X:
- *Tipo de Datos:* ver EJE_X en *MSG*.
- *Tipo de Acceso:* privado
- *Rango:* ver EJE_X en *MSG*.
- *Parte de:* REG_MIMICO
- *Descripción:* ver EJE_X en *MSG*.

1.5 EJE_Y:
- *Tipo de Datos:* ver EJE_Y en *MSG*.
- *Tipo de Acceso:* privado
- *Rango:* ver EJE_Y en *MSG*.
- *Parte de:* REG_MIMICO
- *Descripción:* ver EJE_Y en *MSG*.

1.6 LETRA:
- *Tipo de Datos:* ver LETRA en *MSG*.
- *Tipo de Acceso:* privado
- *Rango:* ver LETRA en *MSG*.
- *Parte de:* REG_MIMICO
- *Descripción:* ver LETRA en *MSG*.

1.7 FONDO:
- *Tipo de Datos:* ver FONDO en *MSG*.
- *Tipo de Acceso:* privado
- *Rango:* ver FONDO en *MSG*.
- *Parte de:* REG_MIMICO
- *Descripción:* ver FONDO en *MSG*.

1.8 LETRA_A:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* REG_MIMICO
- *Descripción:* representa el color de letra con el que se debe escribir el valor del punto de la imagen en estado de alarma.

1.9 FONDO_A:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado

- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Parte de:* REG_MIMICO
- *Descripción:* representa el color de fondo con el que se debe escribir el valor del punto de la imagen en estado de alarma.

2. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_MIMICO
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* cuando se crea un objeto se reserva espacio para n mímicos (elementos de tipo REG_MIMICO). V es el puntero a este espacio de memoria.

3. PROC:
- *Tipo de Datos:* puntero a objeto de tipo *PROCESO*
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* representa un puntero al objeto de tipo *PROCESO* que almacena los registros de procesos en IGNATIUS.

4. VENT:
- *Tipo de Datos:* handler de window HWND
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* representa el handler de la window del supervisor a la que se le enviarán los mensajes (ej.: mensaje para abrir el pipe de comunicaciones, etc.).

5. ISZPIPENAME:
- *Tipo de Datos:* IString (UICL)
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* representa el nombre del pipe de comunicaciones creado y usado por la clase para comunicarse con el supervisor.

6. PAVENTANA:
- *Tipo de Datos:* puntero a IWindow (UICL)
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* representa un puntero a un objeto de tipo IWindow. Este atributo es usado para la creación del pipe de comunicaciones.

7. PAPIPE:
- *Tipo de Datos:* puntero a objeto de tipo *PIPE*
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* representa el puntero al objeto pipe de comunicaciones.

8. PAMSG:
- *Tipo de Datos:* puntero a objeto de tipo MSG
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos

- *Parte de:* MIMICO
- *Descripción:* representa el puntero al objeto de mensajes usado por el pipe de comunicaciones.

9. CONTINUAR:
- *Tipo de Datos:* boolean
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* es el atributo evaluado por el *Administrador de Mímicos* del motor de ejecución para determinar si debe continuar o debe finalizar su ejecución.

10. CANT_ MIMICOS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* representa la cantidad de representaciones almacenadas en el objeto.

11. LONG_TABLA
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* MIMICO
- *Descripción:* representa la cantidad máxima de representaciones que pueden ser almacenadas por el objeto.

La clase mímico posee los siguientes **métodos**:

1. MIMICO (constructor):
- *Tipo de Acceso:* público
- *Parte de:* MIMICO
- *Descripción:* es el constructor de la clase. Inicializa atributos y reserva espacio para almacenar los mímicos. El espacio reservado para los mímicos es alocado dinámicamente con el formato de un arreglo de *s* elementos de tipo REG_MIMICO. Se puede invocar pasando como parámetro un número entero *s* para indicar la cantidad de mímicos y relaciones que el objeto va a almacenar como máximo; en caso contrario, se reserva espacio para un máximo de 10 mímicos.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*:

  s:
  - *E/S:* entrada
  - *Tipo de Datos:* entero
  - *Rango:* cualquier valor positivo representable por el tipo de datos
  - *Descripción:* se utiliza para indicar la cantidad de mímicos que debe almacenar el objeto como máximo.

2. ~MIMICO (destructor):
- *Tipo de Acceso:* público
- *Parte de:* MIMICO
- *Descripción:* es el destructor de la clase. Libera el espacio alocado por el constructor de la clase.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_MIMICO:
- *Tipo de Acceso:* público
- *Parte de:* MIMICO

- *Descripción:* si hay espacio en el arreglo, agrega un registro con los valores provistos como parámetro.
- *Devuelve:* si puede agregar devuelve el índice del arreglo donde agregó el elemento, si NO hay más espacio para agregar devuelve - 1
- *Parámetros*:

tabla:
- *E/S:* entrada
- *Tipo de Datos:* ver *TABLA* en *Atributos*.
- *Rango:* ver *TABLA* en *Atributos*.
- *Descripción:* ver *TABLA* en *Atributos*.

índice:
- *E/S:* entrada
- *Tipo de Datos:* ver *INDICE* en *Atributos*.
- *Rango:* ver *INDICE* en *Atributos*.
- *Descripción:* ver *INDICE* en *Atributos*.

proc_ind:
- *E/S:* entrada
- *Tipo de Datos:* ver *PROC_IND* en *Atributos*.
- *Rango:* ver *PROC_IND* en *Atributos*.
- *Descripción:* ver *PROC_IND* en *Atributos*.

eje_x:
- *E/S:* entrada
- *Tipo de Datos:* ver *EJE_X* en *Atributos*.
- *Rango:* ver *EJE_X* en *Atributos*.
- *Descripción:* ver *EJE_X* en *Atributos*.

eje_y:
- *E/S:* entrada
- *Tipo de Datos:* ver *EJE_Y* en *Atributos*.
- *Rango:* ver *EJE_Y* en *Atributos*.
- *Descripción:* ver *EJE_Y* en *Atributos*.

letra:
- *E/S:* entrada
- *Tipo de Datos:* ver *LETRA* en *Atributos*.
- *Rango:* ver *LETRA* en *Atributos*.
- *Descripción:* ver *LETRA* en *Atributos*.

fondo:
- *E/S:* entrada
- *Tipo de Datos:* ver *FONDO* en *Atributos*.
- *Rango:* ver *FONDO* en *Atributos*.
- *Descripción:* ver *FONDO* en *Atributos*.

letra_a:
- *E/S:* entrada
- *Tipo de Datos:* ver *LETRA_A* en *Atributos*.
- *Rango:* ver *LETRA_A* en *Atributos*.
- *Descripción:* ver *LETRA_A* en *Atributos*.

fondo_a:
- *E/S:* entrada
- *Tipo de Datos:* ver *FONDO_A* en *Atributos*.
- *Rango:* ver *FONDO_A* en *Atributos*.
- *Descripción:* ver *FONDO_A* en *Atributos*.

4. BORRAR_MIMICO:
- *Tipo de Acceso:* público
- *Parte de:* MIMICO
- *Descripción:* si el índice especificado como parámetro es válido, elimina el elemento del arreglo.

- *Devuelve:* 0 si la operación terminó bien, -1 si la operación terminó mal.
- *Parámetros*:

mim_ind:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Descripción:* representa el índice del elemento que hay que borrar dentro del arreglo

5. LEER_MIMICO:
- *Tipo de Acceso:* público
- *Parte de:* MIMICO
- *Descripción:* si el índice especificado como parámetro es válido, devuelve los valores del elemento leído en los parámetros de salida.
- *Devuelve:* 0 si la operación terminó bien, -1 si la operación terminó mal.
- *Parámetros*:

mim_ind:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Descripción:* representa el índice del elemento que hay que leer dentro del arreglo

tabla:
- *E/S:* salida
- *Tipo de Datos:* ver *TABLA* en *Atributos*.
- *Rango:* ver *TABLA* en *Atributos*.
- *Descripción:* ver *TABLA* en *Atributos*.

índice:
- *E/S:* salida
- *Tipo de Datos:* ver *INDICE* en *Atributos*.
- *Rango:* ver *INDICE* en *Atributos*.
- *Descripción:* ver *INDICE* en *Atributos*.

proc_ind:
- *E/S:* salida
- *Tipo de Datos:* ver *PROC_IND* en *Atributos*.
- *Rango:* ver *PROC_IND* en *Atributos*.
- *Descripción:* ver *PROC_IND* en *Atributos*.

eje_x:
- *E/S:* salida
- *Tipo de Datos:* ver *EJE_X* en *Atributos*.
- *Rango:* ver *EJE_X* en *Atributos*.
- *Descripción:* ver *EJE_X* en *Atributos*.

eje_y:
- *E/S:* salida
- *Tipo de Datos:* ver *EJE_Y* en *Atributos*.
- *Rango:* ver *EJE_Y* en *Atributos*.
- *Descripción:* ver *EJE_Y* en *Atributos*.

letra:
- *E/S:* salida
- *Tipo de Datos:* ver *LETRA* en *Atributos*.
- *Rango:* ver *LETRA* en *Atributos*.
- *Descripción:* ver *LETRA* en *Atributos*.

fondo:
- *E/S:* salida
- *Tipo de Datos:* ver *FONDO* en *Atributos*.
- *Rango:* ver *FONDO* en *Atributos*.
- *Descripción:* ver *FONDO* en *Atributos*.

letra_a:
- *E/S:* salida
- *Tipo de Datos:* ver *LETRA_A* en *Atributos.*
- *Rango:* ver *LETRA_A* en *Atributos.*
- *Descripción:* ver *LETRA_A* en *Atributos.*

fondo_a:
- *E/S:* salida
- *Tipo de Datos:* ver *FONDO_A* en *Atributos.*
- *Rango:* ver *FONDO_A* en *Atributos.*
- *Descripción:* ver *FONDO_A* en *Atributos.*

6. BUSCAR_MIMICO:
- *Tipo de Acceso:* público
- *Parte de:* MIMICO
- *Descripción:* recorre secuencialmente el arreglo en busca de un registro cuyos atributos coincidan con los valores pasados como parámetro *((v[i].tabla == tabla) && (v[i].indice == indice) && (v[i].proc_ind == proc_ind)).*
- *Devuelve:* el índice del arreglo donde encontró el elemento, o -1 si no pudo encontrar el elemento buscado.
- *Parámetros*:

tabla:
- *E/S:* entrada
- *Tipo de Datos:* ver *TABLA* en *Atributos.*
- *Rango:* ver *TABLA* en *Atributos.*
- *Descripción:* ver *TABLA* en *Atributos.*

índice:
- *E/S:* entrada
- *Tipo de Datos:* ver *INDICE* en *Atributos.*
- *Rango:* ver *INDICE* en *Atributos.*
- *Descripción:* ver *INDICE* en *Atributos.*

proc_ind:
- *E/S:* entrada
- *Tipo de Datos:* ver *PROC_IND* en *Atributos.*
- *Rango:* ver *PROC_IND* en *Atributos.*
- *Descripción:* ver *PROC_IND* en *Atributos.*

7. DISPLAY:
- *Tipo de Acceso:* público virtual
- *Parte de:* MIMICO
- *Descripción:* este método es el *Administrador de Mímicos* del *Motor de Ejecución*. Es lanzado en un thread independiente de ejecución por el método *LANZAR_MIMICO*. Ejecuta en un loop indefinidamente evaluando en cada ciclo el atributo booleano *CONTINUAR* para determinar si debe seguir ejecutando o debe finalizar la ejecución. Las tareas llevadas a cabo por este método son:

1. Inicialización de variables.
2. Apertura del semáforo de eventos para control de ejecución.
3. Creación de los objetos PIPE y MSG
4. Envío de mensaje al supervisor para que abra el pipe
5. Conexión al pipe
6. Mientras no haya que finalizar la ejecución *(CONTINUAR = TRUE)*

6.1. Bloquearse a la espera de una señal de semáforo de evento
6.2. Evaluar el tiempo transcurrido desde el último ciclo
6.3. Para cada uno de los elementos en el arreglo de mímicos:
6.3.1 Leer el elemento
6.3.2 Si el proceso es el que está mostrándose actualmente *(proc_ind == proc->proceso_en_display)*
6.3.2.1 Leer los datos del proceso

6.3.2.2 Si no hay que refrescar el proceso en pantalla *((tpo_transcurrido + tpo_esp) < frec_disp)* entonces
6.3.2.2.1 Actualizar el tiempo esperado del proceso
6.3.2.3 Si hay que refrescar el proceso en pantalla, entonces
6.3.2.3.1 Leer el valor y el estado del punto de la imagen
6.3.2.3.2 Si el punto está en alarma, entonces setear los valores de color de letra y de fondo correspondientes
6.3.2.3.3 Invocar al método virtual MOSTRAR_PUNTO
6.3.2.3.4 Inicializar el tiempo esperado del proceso
7. Enviar un mensaje al supervisor para que cierre el pipe
- *Devuelve:* no devuelve ningún valor.
- *Parámetros:* no tiene.

8. LANZAR_MIMICO:
- *Tipo de Acceso:* público virtual
- *Parte de:* MIMICO
- *Descripción:* este método está relacionado estrictamente con cuestiones de implementación del *Administrador de Mímicos*. Es por este motivo que fue definido como virtual, para permitir una redefinición en caso de que fuera necesario por cuestiones de implementación (por ej.: este método crea un thread para lanzar el método DISPLAY; los threads son una implementación del OS/2, si se quisiera migrar a otro sistema operativo, el trabajo se minimizaría). Los recursos usados por IGNATIUS que están ligados a la implementación de este caso en particular fueron, en su gran mayoría, identificados y encapsulados dentro de métodos virtuales. Las tareas llevadas a cabo por este método son las siguientes:
1. Inicialización de variables
2. Creación de un semáforo de eventos para el *Planificador de Tareas*.
3. Creación de un thread para la ejecución del método DISPLAY
- *Devuelve:* un puntero a un objeto de tipo IThread (UICL). Este objeto es el thread en el cual ejecuta el método DISPLAY de la clase.
- *Parámetros*

ventana:
- *E/S:* entrada
- *Tipo de Datos:* puntero a IWindow (UICL)
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* este parámetro es usado para la creación del objeto de la clase PIPE y para el envío de mensajes.

proceso:
- *E/S:* entrada
- *Tipo de Datos:* ver PROC en Atributos.
- *Rango:* ver PROC en Atributos.
- *Descripción:* ver PROC en Atributos.

semáforo:
- *E/S:* salida
- *Tipo de Datos:* semáforo de eventos (HEV)
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* representa el valor del manejador del semáforo de eventos usado por la clase TAREA para planificar la ejecución del *Administrador de Mímicos*.

pipename:
- *E/S:* entrada
- *Tipo de Datos:* ver ISZPIPENAME en Atributos.
- *Rango:* ver ISZPIPENAME en Atributos.
- *Descripción:* ver ISZPIPENAME en Atributos.

9 DETENER_MIMICO

- *Tipo de Acceso:* público virtual
- *Parte de:* MIMICO
- *Descripción:* pone en falso el atributo booleano *CONTINUAR* para detener la ejecución del *Administrador de Mímicos.*
- *Devuelve:* no devuelve ningún valor.
- *Parámetros: no tiene.*

10. MOSTRAR_PUNTO:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* MIMICO
- *Descripción:* el código de este método podría ser funcionalmente considerado como parte del código del método *DISPLAY* , aunque fue encapsulado en un método separado para definirlo como virtual y permitir su redefinición en caso de que se quisiera implementar de otra manera. Es protegido ya que sólo puede ser invocado a través del método *DISPLAY* de la clase, porque como se dijo anteriormente, es una misma unidad desde el punto de vista funcional. Las tareas llevadas a cabo son:

1. Instanciar los atributos del mensaje con los valores del punto de la imagen
2. Escribir el mensaje en el pipe
3. Enviar un mensaje al supervisor para decirle que lea un mensaje del pipe y que lo represente por pantalla.

- *Devuelve:* no devuelve ningún valor.
- *Parámetros:*

valor:
- *E/S:* entrada
- *Tipo de Datos:* double
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el valor de el punto de la imagen a representar por pantalla.

eje_x:
- *E/S:* entrada
- *Tipo de Datos:* ver *EJE_X* en *Atributos.*
- *Rango:* ver *EJE_X* en *Atributos.*
- *Descripción:* ver *EJE_X* en *Atributos.*

eje_y:
- *E/S:* entrada
- *Tipo de Datos:* ver *EJE_Y* en *Atributos.*
- *Rango:* ver *EJE_Y* en *Atributos.*
- *Descripción:* ver *EJE_Y* en *Atributos.*

letra:
- *E/S:* entrada
- *Tipo de Datos:* ver *LETRA* en *Atributos.*
- *Rango:* ver *LETRA* en *Atributos.*
- *Descripción:* ver *LETRA* en *Atributos.*

fondo:
- *E/S:* entrada
- *Tipo de Datos:* ver *FONDO* en *Atributos.*
- *Rango:* ver *FONDO* en *Atributos.*
- *Descripción:* ver *FONDO* en *Atributos.*

11 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* MIMICO
- *Descripción:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Devuelve:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Parámetros:* no tiene.

12 SALIRCRITSEC:

- *Tipo de Acceso:* protegido virtual
- *Parte de:* MIMICO
- *Descripción:* ver *SALIRCRITSEC* en *tableaux_int*
- *Devuelve:* ver *SALIRCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.13 CLASE port

La clase *port* maneja los ports de comunicaciones a través de los cuales se intercambian datos entre IGNATIUS y el mundo exterior. También relaciona a los distintos dispositivos físicos con los puntos de la imagen asociados.

La siguiente es la definición de la clase *port* que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class port {
private:
  struct reg_port {
    int direccion, sub_dir;
    HFILE  hFile;
    char descripcion[40];
    char timestamp[20];
  };
  struct reg_port_imagen {
    int port_ind, indice;
    tableaux* tabla;
  };
  reg_port* v;
  reg_port_imagen* w;
  cola*  p_c_in;
  cola*  p_c_out;
  Boolean continuar;
public:
  int cant_ports, cant_relac, long_tabla;
  enum eDisp
{CON,COM1,COM2,COM3,COM4,PRN,LPT1,LPT2,LPT3,NUL,SCREEN$,KBD$,CLOCK$,MOUSE$,POINTER$};
  port(int s=10);
  ~port();
  int agregar_port(int direccion,char* descripcion);
  int modificar_port(int port_ind,int direccion,char* descripcion);
  int leer_port(int port_ind,int& direccion,char* descripcion);
  int leer_relac(int relid,int& port_ind,tableaux*& tabla,int& indice);
  // relacion m-a-m port_imagen
  int relacionar_port_imagen(int port_ind,tableaux* tabla,int indice);
  int buscar_port(tableaux* tabla,int indice);
  void mostrar_port();
  void mostrar_relac();
  int borrar_port(int port_ind);
  int borrar_relac(int relid);
  int abrir_port(int port_ind);
  int cerrar_port(int port_ind);
  int leer_byte(int port_ind,BYTE* pByte);
  int escribir_byte(int port_ind,BYTE byte);
  virtual int leer_string(int port_ind,char* cadena);
  virtual int escribir_string(int port_ind,char* cadena);
  int get_baud_rate(int port_ind,USHORT& bRate);
  int set_baud_rate(int port_ind,USHORT bRate);
  int get_comm_status(int port_ind,UCHAR& ucStatus,ULONG& ulStatusLen);
  int query_in_queue(int port_ind,struct_cola& cIn,ULONG& ulInCount);
  int query_out_queue(int port_ind,struct_cola& cOut,ULONG& ulOutCount);
  int get_dcb_info(int port_ind,struct_dcb& dcb,ULONG& ulDcb);
  int set_dcb_info(int port_ind,struct_dcb dcb,ULONG& ulDcb);
  int get_line_ctrl(int port_ind,struct_lcb& lcb,ULONG& ulLcb);
  int set_line_ctrl(int port_ind,struct_lcb2 lcb,ULONG& ulLcb);
  virtual void manejador();
```

```
  virtual IThread* lanzar_manejador(cola* p_cola_in,cola* p_cola_out,HEV& semaforo);
  virtual void detener_manejador();
  virtual char* nombre_disp(int direccion);
protected:
  virtual int leer_handle(int port_ind,HFILE& hFile);
  virtual int modificar_handle(int port_ind,HFILE hFile);
  virtual unsigned long EntrarCritSec();
  virtual unsigned long SalirCritSec();
};
```

La clase *port* posee los siguientes **atributos**:

1. REG_PORT:

- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* PORT
- *Descripción:* representa la estructura de representación de un port de comunicaciones.

1.1 DIRECCION:

- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* 0..14
- *Parte de:* REG_PORT
- *Descripción:* representa la dirección física del port (por ej.: 0=CON, 1=COM1).

1.2 SUB_DIR:

- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_PORT
- *Descripción:* representa la dirección física del dispositivo

1.3 HFILE:

- *Tipo de Datos:* HFILE
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_PORT
- *Descripción:* representa el handler de archivo del port usado por las funciones de manejo del port.

1.4 DESCRIPCION:

- *Tipo de Datos:* caracter de 40 posiciones
- *Tipo de Acceso:* privado
- *Rango:* cualquier cadena de caracteres de 12 posiciones que finalice con el caracter de fin de cadena
- *Parte de:* REG_PORT
- *Descripción:* representa la descripción del registro de port.

1.5 TIMESTAMP:

- *Tipo de Datos:* caracter de 20 posiciones
- *Tipo de Acceso:* privado
- *Rango:* ver *TIMESTAMP* en *tableaux_int.*
- *Parte de:* REG_PORT
- *Descripción:* representa la fecha y hora en la cual fue modificado por última vez algún valor del registro de port.

2. REG_PORT_IMAGEN:

- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* PORT
- *Descripción:* representa la relación entre los ports de comunicaciones y los puntos de la imagen.

2.1 PORT_IND:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_PORT_IMAGEN
- *Descripción:* representa el índice del port relacionado.

2.2 INDICE:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* REG_PORT_IMAGEN
- *Descripción:* representa el índice del punto de la imagen relacionado.

2.3 TABLA:
- *Tipo de Datos:* puntero a TABLEAUX
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_PORT_IMAGEN
- *Descripción:* representa el puntero al segmento virtual de la imagen relacionado.

3. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_PORT
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* PORT
- *Descripción:* cuando se crea un objeto se reserva espacio para n ports (elementos de tipo REG_PORT). V es el puntero a este espacio de memoria.

4. W:
- *Tipo de Datos:* puntero a estructura de tipo REG_PORT_IMAGEN
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* PORT
- *Descripción:* cuando se crea un objeto se reserva espacio para n relaciones port_imagen (elementos de tipo REG_PORT_IMAGEN). W es el puntero a este espacio de memoria.

5. P_C_IN:
- *Tipo de Datos:* puntero a objeto COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* PORT
- *Descripción:* es un puntero al objeto que representa la cola de entrada. Esta cola es procesada por el *Manejador de Ports* del motor de ejecución.

6. P_C_OUT:
- *Tipo de Datos:* puntero a objeto COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.

- *Parte de:* PORT
- *Descripción:* es un puntero al objeto que representa la cola de salida. Esta cola es procesada por el *Manejador de Ports* del motor de ejecución.

7. CONTINUAR:
- *Tipo de Datos:* boolean
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* PORT
- *Descripción:* es el atributo evaluado por el *Manejador de Ports* del motor de ejecución para determinar si debe continuar o debe finalizar su ejecución.

8. CANT_PORTS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* PORT
- *Descripción:* representa la cantidad de ports de comunicaciones existentes en el objeto.

9. CANT_RELAC:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* PORT
- *Descripción:* representa la cantidad de relaciones port-imagen existentes en el objeto.

10. LONG_TABLA:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* PORT
- *Descripción:* representa la cantidad máxima de ports de comunicaciones que puede almacenar el objeto.

11. EDISP:
- *Tipo de Datos:* enumerado
- *Tipo de Acceso:* público
- *Rango:* CON, COM1, COM2, COM3, COM4, PRN, LPT1, LPT2, LPT3, NUL, SCREEN$, KBD$, CLOCK$, MOUSE$, POINTER$
- *Parte de:* PORT
- *Descripción:* representa la dirección del port de comunicaciones.

La clase mímico posee los siguientes **métodos:**

1. PORT (constructor):
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* es el constructor de la clase. Reserva espacio para almacenar los datos de los ports de comunicaciones y sus relaciones con los puntos de la imagen; también inicializa los valores de los atributos. El espacio reservado para los ports es alocado dinámicamente con el formato de un arreglo de *s* elementos de tipo REG_PORT. El espacio reservado para las relaciones es alocado dinámicamente con el formato de un arreglo de *s*3* elementos de tipo REG_PORT_IMAGEN. Se puede invocar pasando como parámetro un número entero *s* para indicar la cantidad de ports y relaciones que el objeto va a almacenar como máximo; en caso contrario, se reserva espacio para un máximo de 10 ports y 30 relaciones.

-Devuelve: no devuelve ningún valor

- Parámetros:

s:

- E/S: entrada

- Tipo de Datos: entero

- Rango: cualquier valor positivo representable por el tipo de datos

- Descripción: se utiliza para indicar la cantidad de ports que debe almacenar el objeto como máximo.

2. ~PORT (destructor):

- Tipo de Acceso: público

- Parte de: PORT

- Descripción: es el destructor de la clase. Libera el espacio alocado por el constructor de la clase.

- Devuelve: no devuelve ningún valor

- Parámetros: no tiene

3. AGREGAR_PORT:

- Tipo de Acceso: público

- Parte de: PORT

- Descripción: si hay espacio en el arreglo, agrega un registro con los valores provistos como parámetro.

- Devuelve: si puede agregar devuelve el índice del arreglo donde agregó el elemento, si NO hay más espacio para agregar devuelve - 1

- Parámetros:

dirección:

- E/S: entrada

- Tipo de Datos: ver *DIRECCION* en *Atributos*.

- Rango: ver *DIRECCION* en *Atributos*.

- Descripción: ver *DIRECCION* en *Atributos*.

sub_dir:

- E/S: entrada

- Tipo de Datos: ver *SUB_DIR* en *Atributos*.

- Rango: ver *SUB_DIR* en *Atributos*.

- Descripción: ver *SUB_DIR* en *Atributos*.

descripción:

- E/S: entrada

- Tipo de Datos: puntero a caracter

- Rango: ver *DESCRIPCION* en *Atributos*.

- Descripción: ver *DESCRIPCION* en *Atributos*.

4. MODIFICAR_PORT:

- Tipo de Acceso: público

- Parte de: PORT

- Descripción: si el índice pasado como parámetro es válido, modifica el registro de port con los valores provistos como parámetros.

- Devuelve: 0 si puede realizar la operación, -1 si no puede realizar la operación.

- Parámetros:

port_ind:

- E/S: entrada

- Tipo de Datos: entero

- Rango: cualquier valor representable por el tipo de datos.

- Descripción: representa la posición dentro del arreglo del elemento a modificar.

direccion: ver AGREGAR_PORT

sub_dir: ver AGREGAR_PORT

descripción: ver AGREGAR_PORT

5. LEER_PORT:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, devuelve los valores del elemento leído en las variables provistas como parámetros.
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:

port_ind:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* representa la posición dentro del arreglo del elemento a leer.

direccion:
- *E/S:* salida
- *Tipo de Datos:* ver *DIRECCION* en *Atributos*.
- *Rango:* ver *DIRECCION* en *Atributos*.
- *Descripción:* ver *DIRECCION* en *Atributos*.

sub_dir:
- *E/S:* salida
- *Tipo de Datos:* ver *SUB_DIR* en *Atributos*.
- *Rango:* ver *SUB_DIR* en *Atributos*.
- *Descripción:* ver *SUB_DIR* en *Atributos*.

descripción:
- *E/S:* salida
- *Tipo de Datos:* puntero a caracter
- *Rango:* ver *DESCRIPCION* en *Atributos*.
- *Descripción:* ver *DESCRIPCION* en *Atributos*.

6. LEER_RELAC:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, devuelve los valores del elemento de relación leído en las variables provistas como parámetros.
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:

relid:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* representa la posición dentro del arreglo de relación del elemento a leer.

port_ind:
- *E/S:* salida
- *Tipo de Datos:* ver *PORT_IND* en *Atributos*
- *Rango:* ver *PORT_IND* en *Atributos*
- *Descripción:* ver *PORT_IND* en *Atributos*

tabla:
- *E/S:* salida
- *Tipo de Datos:* ver *TABLA* en *Atributos*.
- *Rango:* ver *TABLA* en *Atributos*.
- *Descripción:* ver *TABLA* en *Atributos*.

índice:
- *E/S:* salida
- *Tipo de Datos:* ver *INDICE* en *Atributos*.

- *Rango:* ver *INDICE* en *Atributos.*
- *Descripción:* ver *INDICE* en *Atributos.*

7. RELACIONAR_PORT_IMAGEN:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si hay espacio en el arreglo, agrega un registro de relación con los valores provistos como parámetro.
- *Devuelve:* si puede agregar devuelve el índice del arreglo donde agregó el elemento, si NO hay más espacio para agregar devuelve - 1
- *Parámetros*:

port_ind:
- *E/S:* entrada
- *Tipo de Datos:* ver *PORT_IND* en *Atributos*
- *Rango:* ver *PORT_IND* en *Atributos*
- *Descripción:* ver *PORT_IND* en *Atributos*

tabla:
- *E/S:* entrada
- *Tipo de Datos:* ver *TABLA* en *Atributos.*
- *Rango:* ver *TABLA* en *Atributos.*
- *Descripción:* ver *TABLA* en *Atributos.*

índice:
- *E/S:* entrada
- *Tipo de Datos:* ver *INDICE* en *Atributos.*
- *Rango:* ver *INDICE* en *Atributos.*
- *Descripción:* ver *INDICE* en *Atributos.*

8. BUSCAR_PORT:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* recorre el arreglo de relación buscando si existe algún port relacionado al punto de la imagen pasado como parámetro *((w[i].tabla == tabla) && (w[i].índice == índice)).*
- *Devuelve:* el port_ind del port relacionado al punto de la imagen pasado como parámetro, o -1 en el caso de no encontrar ningún port relacionado.
- *Parámetros*:

tabla: ver *RELACIONAR_PORT_IMAGEN*
índice: ver *RELACIONAR_PORT_IMAGEN*

9. BORRAR_PORT:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, borra del arreglo el elemento especificado.
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:

port_ind: ver *RELACIONAR_PORT_IMAGEN*

10. BORRAR_RELAC:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, borra del arreglo el elemento de relación especificado.
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:

relid: ver *LEER_RELAC*

11. ABRIR_PORT:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, abre el port especificado para lectura y escritura. Encapsula la instrucción *DosOpen* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:
  port_ind: ver *RELACIONAR_PORT_IMAGEN*

12. CERRAR_PORT:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, cierra el port especificado. Encapsula la instrucción *DosClose* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:
  port_ind: ver *RELACIONAR_PORT_IMAGEN*

13. LEER_BYTE:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, lee un byte del port especificado. Encapsula la instrucción *DosRead* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:
  port_ind: ver *RELACIONAR_PORT_IMAGEN*
  pbyte:
  - *E/S:* salida
  - *Tipo de Datos:* puntero a BYTE
  - *Rango:* cualquier valor representable por el tipo de datos
  - *Descripción:* puntero al byte leído del port especificado.

14. ESCRIBIR_BYTE:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, escribe un byte en el port especificado. Encapsula la instrucción *DosWrite* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:
  port_ind: ver *RELACIONAR_PORT_IMAGEN*
  byte:
  - *E/S:* salida
  - *Tipo de Datos:* BYTE
  - *Rango:* cualquier valor representable por el tipo de datos
  - *Descripción:* representa el byte que se debe escribir en el port especificado.

15. LEER_STRING:
- *Tipo de Acceso:* público virtual
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, lee una cadena de caracteres del port especificado. Encapsula la instrucción *DosRead* [ver OS/2 Control Program Guide and Reference]. Fue definido como virtual para permitir su redefinición por cuestiones de implementación.
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:
  port_ind: ver *RELACIONAR_PORT_IMAGEN*
  cadena:

- *E/S:* salida
- *Tipo de Datos:* puntero a cadena de caracteres
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* puntero a la cadena de caracteres leída del port especificado.

16. ESCRIBIR_STRING:
- *Tipo de Acceso:* público virtual
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, escribe una cadena de caracteres en el port especificado. Encapsula la instrucción *DosWrite* [ver OS/2 Control Program Guide and Reference]. Fue definido como virtual para permitir su redefinición por cuestiones de implementación.
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros*:

port_ind: ver *RELACIONAR_PORT_IMAGEN*
cadena:
- *E/S:* entrada
- *Tipo de Datos:* puntero a cadena de caracteres
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* puntero a la cadena de caracteres que se debe escribir en el port especificado.

17. GET_BAUD_RATE:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, devuelve en el parámetro *brate* el baud rate del port especificado. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*

0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial

- *Parámetros*:

port_ind: ver *RELACIONAR_PORT_IMAGEN*
brate:
- *E/S:* salida
- *Tipo de Datos:* USHORT
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el baud rate del port especificado.

18. SET_BAUD_RATE:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, setea el baud rate del port especificado con el valor del parámetro *brate*. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*

0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial

- *Parámetros*:

1. port_ind: ver *RELACIONAR_PORT_IMAGEN*
2. brate:
- *E/S:* entrada
- *Tipo de Datos:* USHORT
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el baud rate del port especificado.

19. GET_COMM_STATUS:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, devuelve en los parámetros de salida el status del port especificado. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*
0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial
- *Parámetros:*
port_ind: ver *RELACIONAR_PORT_IMAGEN*
ucstatus:
- *E/S:* salida
- *Tipo de Datos:* UCHAR
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la información de status del port especificado.
ulstatuslen:
- *E/S:* salida
- *Tipo de Datos:* ULONG
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la longitud en bytes del parámetro *ucstatus*.

20. QUERY_IN_QUEUE:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, devuelve en los parámetros de salida los valores de la cola de entrada del port especificado. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*
0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial
- *Parámetros:*
port_ind: ver *RELACIONAR_PORT_IMAGEN*
cin:
- *E/S:* salida
- *Tipo de Datos:* STRUCT_COLA
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la estructura de información de la cola de entrada del port especificado.
ulincount:
- *E/S:* salida
- *Tipo de Datos:* ULONG
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la longitud en bytes del parámetro *cin*.

21. QUERY_OUT_QUEUE:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, devuelve en los parámetros de salida los valores de la cola de salida del port especificado. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*
0: Si puede realizar la operación con éxito
-1: Si la operación termina con error

-2: Si el port NO es Serial
- *Parámetros*:
port_ind: ver *RELACIONAR_PORT_IMAGEN*
cout:
- *E/S:* salida
- *Tipo de Datos:* STRUCT_COLA
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la estructura de información de la cola de salida del port especificado.
uloutcount:
- *E/S:* salida
- *Tipo de Datos:* ULONG
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la longitud en bytes del parámetro *cout*.

22. GET_DCB_INFO:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, devuelve en los parámetros de salida los valores del *Device Control Block* del port especificado. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*
0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial
- *Parámetros*:
port_ind: ver *RELACIONAR_PORT_IMAGEN*
dcb:
- *E/S:* salida
- *Tipo de Datos:* STRUCT_DCB
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la estructura de información del *Device Control Block* del port especificado
uldcb:
- *E/S:* salida
- *Tipo de Datos:* ULONG
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la longitud en bytes del parámetro *dcb*.

23. SET_DCB_INFO:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, setea los valores del *Device Control Block* del port especificado. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*
0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial
- *Parámetros*:
port_ind: ver *RELACIONAR_PORT_IMAGEN*
dcb:
- *E/S:* entrada
- *Tipo de Datos:* STRUCT_DCB
- *Rango:* cualquier valor representable por el tipo de datos

- *Descripción:* representa la estructura de información del *Device Control Block* del port especificado

uldcb:
- *E/S:* salida
- *Tipo de Datos:* ULONG
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la longitud en bytes del parámetro *dcb*.

24. GET_LINE_CTRL:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, devuelve en los parámetros de salida información de las características de la línea del port especificado. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*

0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial

- *Parámetros*:

port_ind: ver *RELACIONAR_PORT_IMAGEN*

lcb:
- *E/S:* salida
- *Tipo de Datos:* STRUCT_LCB
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la estructura de información de las características de la línea del port especificado

ullcb:
- *E/S:* salida
- *Tipo de Datos:* ULONG
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la longitud en bytes del parámetro *lcb*.

25. SET_LINE_CTRL:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, setea las características de la línea del port especificado con los valores especificados en los parámetros de entrada. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*

0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial

- *Parámetros*:

port_ind: ver *RELACIONAR_PORT_IMAGEN*

lcb:
- *E/S:* entrada
- *Tipo de Datos:* STRUCT_LCB2
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la estructura de información de las características de la línea del port especificado

ullcb:
- *E/S:* salida
- *Tipo de Datos:* ULONG
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la longitud en bytes del parámetro *lcb*.

26. SET_MODEM_CTRL:
- *Tipo de Acceso:* público
- *Parte de:* PORT
- *Descripción:* si el índice pasado como parámetro es válido, setea las señales DTR y RTS del módem con los valores especificados en los parámetros de entrada. Encapsula la instrucción *DosDevIOCtl* [ver OS/2 Control Program Guide and Reference].
- *Devuelve:*
0: Si puede realizar la operación con éxito
-1: Si la operación termina con error
-2: Si el port NO es Serial
- *Parámetros*:
port_ind: ver *RELACIONAR_PORT_IMAGEN*
mcb:
- *E/S:* entrada
- *Tipo de Datos:* STRUCT_MCB
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la estructura de información de las señales DTR y RTS del módem.
ulmcb:
- *E/S:* salida
- *Tipo de Datos:* ULONG
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa la longitud en bytes del parámetro *mcb*.

27. MANEJADOR:
- *Tipo de Acceso:* público virtual
- *Parte de:* PORT
- *Descripción:* este método es el *Manejador de Ports* del *Motor de Ejecución*. Es lanzado en un thread independiente de ejecución por el método *LANZAR_MANEJADOR*. Ejecuta en un loop indefinidamente evaluando en cada ciclo el atributo booleano *CONTINUAR* para determinar si debe seguir ejecutando o debe finalizar la ejecución. Las tareas llevadas a cabo dentro del ciclo son las siguientes:
1. Bloquearse a la espera de una señal de semáforo de evento
2. Mientras haya elementos en la cola de salida:
2.1 Sacar el elemento de la cola
2.2 Escribir el elemento en todos los ports relacionados al punto de la imagen
3. Para cada uno de los ports:
3.1 Leer el port si es que tiene algo
3.1.1 Si puede leer, usa el byte leído para armar el mensaje
3.2 Escribir lo leído en cada uno de los puntos de la imagen relacionados a ese port usando la cola de input
- *Devuelve:* no devuelve ningún valor.
- *Parámetros:* no tiene.

28 LANZAR_MANEJADOR:
- *Tipo de Acceso:* público virtual
- *Parte de:* PORT
- *Descripción:* este método esta relacionada estrictamente con cuestiones de implementación del *Manejador de Ports*. Es por este motivo que fue definido como virtual, para permitir una redefinición en caso de que fuera necesario por cuestiones de implementación (por ej.: este método crea un thread para lanzar el método MANEJADOR; los threads son una implementación del OS/2, si se quisiera migrar a otro sistema operativo, el trabajo se minimizaría). Los recursos usados por IGNATIUS que están ligados a la implementación de este caso en particular fueron, en su gran mayoría, identificados y encapsulados dentro de métodos virtuales. Las tareas llevadas a cabo por este método son las siguientes:

1. Inicialización de variables
2. Creación de un semáforo de eventos para el *Planificador de Tareas.*
3. Creación de un thread para la ejecución del método MANEJADOR

- *Devuelve:* un puntero a un objeto de tipo IThread (UICL). Este objeto es el thread en el cual ejecuta el método MANEJADOR de la clase.
- *Parámetros*

p_cola_in:
- *E/S:* entrada
- *Tipo de Datos:* puntero a objeto de tipo COLA
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* puntero al objeto que representa la cola de entrada de IGNATIUS.

p_cola_out:
- *E/S:* entrada
- *Tipo de Datos:* puntero a objeto de tipo COLA
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* puntero al objeto que representa la cola de salida de IGNATIUS.

semáforo:
- *E/S:* salida
- *Tipo de Datos:* semáforo de eventos (HEV)
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* representa el valor del manejador del semáforo de eventos usado por la clase TAREA para planificar la ejecución del *Analizador de Alarmas.*

29 DETENER_MANEJADOR:
- *Tipo de Acceso:* público virtual
- *Parte de:* PORT
- *Descripción:* pone en falso el atributo booleano *CONTINUAR* para detener la ejecución del *Manejador de Ports.*
- *Devuelve:* no devuelve ningún valor.
- *Parámetros: no tiene.*

30 NOMBRE_DISP:
- *Tipo de Acceso:* público virtual
- *Parte de:* PORT
- *Descripción:* en función de la dirección del dispositivo pasada como parámetro, devuelve el nombre físico del mismo.
- *Devuelve:* un puntero a un string con el nombre físico del dispositivo
- *Parámetros:*

direccion:
- *E/S:* entrada
- *Tipo de Datos:* ver *DIRECCION* en *Atributos.*
- *Rango:* ver *DIRECCION* en *Atributos.*
- *Descripción:* ver *DIRECCION* en *Atributos.*

31 LEER_HANDLE:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* PORT
- *Descripción:* devuelve en el parámetro de salida *hfile* el handler de archivo del dispositivo especificado en el parámetro *port_ind.*
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros:*

port_ind: ver *RELACIONAR_PORT_IMAGEN*

hfile:
- *E/S:* salida
- *Tipo de Datos:* ver *HFILE* en *Atributos.*

- *Rango:* ver *HFILE* en *Atributos.*
- *Descripción:* ver *HFILE* en *Atributos.*

32 MODIFICAR_HANDLE:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* PORT
- *Descripción:* modifica el handle de archivo del dispositivo especificado en el parámetro *port_ind* con el valor especificado en el parámetro *hfile.*.
- *Devuelve:* 0 si puede realizar la operación, -1 si no puede realizar la operación.
- *Parámetros:*

  port_ind: ver *RELACIONAR_PORT_IMAGEN*

  hfile:
  - *E/S:* entrada
  - *Tipo de Datos:* ver *HFILE* en *Atributos.*
  - *Rango:* ver *HFILE* en *Atributos.*
  - *Descripción:* ver *HFILE* en *Atributos.*

33 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* PORT
- *Descripción:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Devuelve:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

34 SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* PORT
- *Descripción:* ver *SALIRCRITSEC* en *tableaux_int*
- *Devuelve:* ver *SALIRCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.14 CLASE comando

La clase *comando* provee al usuario un mecanismo para definir y administrar los comandos ingresados por el operador. Además posee un mecanismo de validación que permite determinar si un comando ingresado por el operador es válido y si los parámetros especificados son correctos.

Si bien el objetivo de la clase no es proveer al usuario de IGNATIUS un conjunto de comandos, sino una herramienta que facilite la definición y administración de los comandos definidos por el usuario, se proveen dos comandos embebidas en la clase a saber:

1. FRECUENCIA_DISPLAY: modifica la frecuencia de display de un proceso especificado. Ejemplo de invocación de comando: **frecuencia_display,1,800**. Esto quiere decir que cada 800 centisegundos se va a actualizar por pantalla los valores del proceso cuyo índice es el 1.
*2. PROCESO_EN_DISPLAY: activa en el display el proceso especificado como parámetro Ejemplo de invocación de comando: **proceso_en_display,1**. Esto quiere decir que a partir del momento en que se ingresó el comando, se mostrará en la pantalla el proceso cuyo índice es el 1.

El *Intérprete de Comandos* espera encontrar a el comando y los argumentos separados por comas. (Por ej.: **frecuencia_display,1,800).**

La siguiente es la definición de la clase comando que se encuentra dentro del archivo header de la biblioteca (ver USANDO LA BIBLIOTECA):

```
class comando {
  struct reg_comando {
    char        cmd[30];
    PFUNCION    p_fcn;
    char        mask[50];
  };
  reg_comando* v;
  p_objetos* p_p_objetos;
  struct reg_cola {
    char cmd_parms[80];
    void* p_algo;
    char timestamp[20];
    reg_cola* proximo;
  };
  reg_cola* primero;
  reg_cola* ultimo;
  reg_cola* auxiliar;
  Boolean continuar;
public:
  int cant_comandos, long_tabla;
  comando(unsigned int s,p_objetos* pp_obj);
  ~comando();
  int agregar_comando(char* comando,PFUNCION p_funcion,char* mask);
  int borrar_comando(int cmd_ind);
  int leer_comando(int cmd_ind,char* comando,PFUNCION& p_funcion,char* mask);
  int buscar_funcion(char* comando,PFUNCION& p_funcion);
  void mostrar_comandos();
  int ejecutar_comando_inm(char* cadena,void* p_algo);
  int agregar_comando_cola(char* cadena,void* p_algo);
  void listar_cola();
  virtual void ejecutor();
  virtual IThread* lanzar_ejecutor(HEV& semaforo);
  virtual void detener_ejecutor();
protected:
  int sacar_comando_cola(char*& cadena,void*& p_algo);
  Boolean existe_comando(char* comando);
  int validar_parametros(char* comando,char* parametros);
  int que_indice(char* comando);
  virtual unsigned long EntrarCritSec();
  virtual unsigned long SalirCritSec();
};
```

La clase comando posee los siguientes **atributos**:

1. REG_COMANDO:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* COMANDO
- *Descripción:* representa la estructura de representación de los comandos.

1.1 CMD:
- *Tipo de Datos:* char[30]
- *Tipo de Acceso:* privado
- *Rango:* cualquier cadena de caracteres de hasta 30 posiciones cuyo último caracter sea el de fin de cadena.
- *Parte de:* REG_COMANDO
- *Descripción:* representa el nombre del comando.

1.2 P_FCN:
- *Tipo de Datos:* PFUNCION
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_COMANDO

- *Descripción:* representa el puntero a la rutina de atención del comando.

1.3 MASK:
- *Tipo de Datos:* char[50]
- *Tipo de Acceso:* privado
- *Rango:* cualquier cadena de caracteres de hasta 50 posiciones cuyo último caracter sea el de fin de cadena.
Los caracteres de esta cadena pueden ser:
i o I: chequea que el argumento sea un número entero.
d o D: chequea que el argumento sea un número que puede tener punto decimal.
Cualquier otro caracter: no realiza ningún chequeo del argumento.
- *Parte de:* REG_COMANDO
- *Descripción:* representa la cantidad, secuencia y el tipo de los argumentos que espera recibir el comando.

2. V:
- *Tipo de Datos:* puntero a estructura de tipo REG_COMANDO
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* COMANDO
- *Descripción:* cuando se crea un objeto se reserva espacio para n comandos (elementos de tipo REG_COMANDO). V es el puntero a este espacio de memoria.

3. P_P_OBJETOS:
- *Tipo de Datos:* puntero a estructura de tipo P_OBJETOS
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* COMANDO
- *Descripción:* representa el puntero a la estructura que contiene los punteros a todos los objetos de IGNATIUS.

4. REG_COLA:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* privado
- *Rango:* no aplica
- *Parte de:* COMANDO
- *Descripción:* representa la estructura de un comando ingresado por el operados. Estos comandos son encadenados en una lista para luego ser procesados por el *Intérprete de Comandos* del motor de ejecución.

4.1 CMD_PARMS:
- *Tipo de Datos:* char[80]
- *Tipo de Acceso:* privado
- *Rango:* cualquier cadena de caracteres de hasta 80 posiciones cuyo último caracter sea el de fin de cadena.
- *Parte de:* REG_COLA
- *Descripción:* representa el comando y los parámetros ingresados por el operador.

4.2 P_ALGO:
- *Tipo de Datos:* puntero a VOID
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* REG_COLA

- *Descripción:* representa un puntero a cualquier objeto o dato. Puede ser utilizado por el supervisor para pasar cualquier información que considere de utilidad a la rutina de atención de comandos.

4.3 TIMESTAMP:
- *Tipo de Datos:* caracter de 20 posiciones
- *Tipo de Acceso:* privado
- *Rango:* ver *TIMESTAMP* en *tableaux_int.*
- *Parte de:* REG_COLA
- *Descripción:* representa la fecha y hora en la cual fue agregado el comando a la lista encadenada.

4.4 PROXIMO:
- *Tipo de Datos:* puntero a REG_COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* REG_COLA
- *Descripción:* representa el puntero al próximo elemento dentro de la lista encadenada

5. PRIMERO:
- *Tipo de Datos:* puntero a REG_COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* COMANDO
- *Descripción:* representa el puntero al primer elemento de la lista encadenada de comandos ingresados por el operador.

6. ULTIMO:
- *Tipo de Datos:* puntero a REG_COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* COMANDO
- *Descripción:* representa el puntero al último elemento de la lista encadenada de comandos ingresados por el operador.

7. AUXILIAR:
- *Tipo de Datos:* puntero a REG_COLA
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* COMANDO
- *Descripción:* representa un puntero auxiliar a un elemento de la lista encadenada de comandos ingresados por el operador.

8. CONTINUAR:
- *Tipo de Datos:* boolean
- *Tipo de Acceso:* privado
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* COMANDO
- *Descripción:* es el atributo evaluado por el *Intérprete de Comandos* del motor de ejecución para determinar si debe continuar o debe finalizar su ejecución.

9. CANT_COMANDOS:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* COMANDO

- *Descripción:* representa la cantidad de comandos existentes en el objeto.

10. LONG_TABLA:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* COMANDO
- *Descripción:* representa la cantidad máxima de comandos que puede almacenar el objeto.

La clase *comando* posee los siguientes **métodos**:

1. COMANDO (constructor):
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* es el constructor de la clase. Reserva espacio para almacenar la definición de comandos y agrega la definición de las funciones básicas provistas por IGNATIUS (FRECUENCIA_DISPLAY y PROCESO_EN_DISPLAY). También inicializa los valores de los atributos. El espacio reservado para los comandos es alocado dinámicamente con el formato de un arreglo de *s* elementos de tipo REG_COMANDO. Se puede invocar pasando como parámetro un número entero *s* para indicar la cantidad de comandos que el objeto va a almacenar como máximo.
- *Devuelve:* no devuelve ningún valor
- *Parámetros*:

  s:
  - *E/S:* entrada
  - *Tipo de Datos:* entero sin signo
  - *Rango:* cualquier valor representable por el tipo de datos
  - *Descripción:* se utiliza para indicar la cantidad de comandos que debe almacenar el objeto como máximo.

  pp_obj:
  - *E/S:* entrada
  - *Tipo de Datos:* ver P_P_OBJETOS en *Atributos*.
  - *Rango:* ver P_P_OBJETOS en *Atributos*.
  - *Descripción:* ver P_P_OBJETOS en *Atributos*.

2. ~COMANDO (destructor):
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* es el destructor de la clase. Libera el espacio alocado por el constructor del objeto y también el. alocado dinámicamente para la cola de comandos a procesar.
- *Devuelve:* no devuelve ningún valor
- *Parámetros: no tiene*

3. AGREGAR_COMANDO:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* si es posible, agrega una definición de comando en el arreglo.
- *Devuelve:*
- *el subíndice del arreglo* donde agregó el elemento, si puede agregar o
- *NO_HAY_ESPACIO* si NO hay más espacio para agregar o
- *COMANDO_YA_EXISTE* si el comando ya existía en la tabla o
- *ERROR_CANTIDAD_ARGUMENTOS* si la long. de parámetros no es igual que la cantidad de parámetros o
- *COMANDO_NO_VALIDO* si el comando es NULL
- *Parámetros:*

  comando:

- *E/S:* entrada
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver NOMBRE en *Atributos.*

p_función:
- *E/S:* entrada
- *Tipo de Datos:* ver P_FCN en *Atributos.*
- *Rango:* ver P_FCN en *Atributos.*
- *Descripción:* ver P_FCN en *Atributos.*

mask:
- *E/S:* entrada
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver MASK en *Atributos.*

4. BORRAR_COMANDO:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* si el índice pasado como parámetro es válido, borra el elemento especificado del arreglo.
- *Devuelve:*
- *OK* si la operación terminó bien,
- *INDICE_INCORRECTO* si no pudo borrar en la tabla
- *Parámetros:*

cmd_ind:
- *E/S:* entrada
- *Tipo de Datos:* entero
- *Rango:* cualquier valor positivo representable por el tipo de datos.
- *Descripción:* representa la posición dentro del arreglo del elemento a borrar.

5. LEER_COMANDO:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* si el índice pasado como parámetro es válido, devuelve en los parámetros de salida los valores del elemento leído.
- *Devuelve:*
- *OK* si la operación terminó bien,
- *INDICE_INCORRECTO* si no pudo leer
- *Parámetros:*

cmd_ind:
- *E/S:* entrada
- *Tipo de Datos:* ver parámetro *cmd_ind* en *BORRAR_COMANDO.*
- *Rango:* ver parámetro *cmd_ind* en *BORRAR_COMANDO.*
- *Descripción:* ver parámetro *cmd_ind* en *BORRAR_COMANDO.*

comando:
- *E/S:* salida
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver NOMBRE en *Atributos.*

p_función:
- *E/S:* salida
- *Tipo de Datos:* ver P_FCN en *Atributos.*
- *Rango:* ver P_FCN en *Atributos.*
- *Descripción:* ver P_FCN en *Atributos.*

mask:

- *E/S:* salida
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver MASK en *Atributos*.

6. BUSCAR_FUNCION:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* recorre secuencialmente el arreglo buscando un elemento cuyo nombre de comando coincida con el argumento de entrada.
- *Devuelve:*
- *OK* si la operación terminó satisfactoriamente
- *COMANDO_INEXISTENTE* si el comando no existe
- *Parámetros:*

comando:
- *E/S:* entrada
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver NOMBRE en *Atributos*.

p_función:
- *E/S:* salida
- *Tipo de Datos:* ver P_FCN en *Atributos*.
- *Rango:* ver P_FCN en *Atributos*.
- *Descripción:* ver P_FCN en *Atributos*.

7. EJECUTAR_COMANDO_INM:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* se valida que el comando y los parámetros estén ok, y se ejecuta la rutina de atención de comando.
- *Devuelve:*
- *OK* si la operación terminó satisfactoriamente
- *COMANDO_INEXISTENTE* si el comando no existe
- *COMANDO_NO_VALIDO* si el comando es NULL
- *Parámetros:*

cadena:
- *E/S:* entrada
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver CMD_PARMS en *Atributos*.

p_algo:
- *E/S:* entrada
- *Tipo de Datos:* puntero a VOID
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver P_ALGO en *Atributos*.

8. AGREGAR_COMANDO_COLA:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* se valida que el comando y los parámetros estén ok, y se agrega un elemento a la cola de comandos ingresados por el operador con los valores provistos como parámetros.
- *Devuelve:*
- *OK* si la operación terminó satisfactoriamente
- *COMANDO_INEXISTENTE* si el comando no existe
- *ERROR_CANTIDAD_ARGUMENTOS* si la cantidad de argumentos no es correcta

- *ERROR_TIPO_ARGUMENTOS* si el tipo de algún argumento no es correcto
- *COMANDO_NO_VALIDO* si el comando es NULL
- *Parámetros:*

cadena: ver *EJECUTAR_COMANDO_INM*
p_algo: ver *EJECUTAR_COMANDO_INM*

9. EJECUTOR:
- *Tipo de Acceso:* público virtual
- *Parte de:* COMANDO
- *Descripción:* este método es el *Intérprete de Comandos* del *Motor de Ejecución*. Es lanzado en un thread independiente de ejecución por el método *LANZAR_EJECUTOR*. Ejecuta en un loop indefinidamente evaluando en cada ciclo el atributo booleano *CONTINUAR* para determinar si debe seguir ejecutando o debe finalizar la ejecución. Las tareas llevadas a cabo dentro del ciclo son las siguientes:

1. Bloquearse a la espera de una señal de semáforo de evento
2. Mientras haya elementos en la cola de comandos ingresados por el operador
2.1 Sacar el primer elemento de la cola de comandos
2.2 Ejecutar inmediatamente el comando (método *EJECUTAR_COMANDO_INM*).

- *Devuelve:* no devuelve ningún valor.
- *Parámetros:* no tiene.

10. LANZAR_EJECUTOR:
- *Tipo de Acceso:* público virtual
- *Parte de:* COMANDO
- *Descripción:* este método esta relacionada estrictamente con cuestiones de implementación del *Intérprete de Comandos*. Es por este motivo que fue definido como virtual, para permitir una redefinición en caso de que fuera necesario por cuestiones de implementación (por ej.: este método crea un thread para lanzar el método EJECUTOR; los threads son una implementación del OS/2, si se quisiera migrar a otro sistema operativo, el trabajo se minimizaría). Los recursos usados por IGNATIUS que están ligados a la implementación de este caso en particular fueron, en su gran mayoría, identificados y encapsulados dentro de métodos virtuales. Las tareas llevadas a cabo por este método son las siguientes:

1. Inicialización de variables
2. Creación de un semáforo de eventos para el *Planificador de Tareas*.
3. Creación de un thread para la ejecución del método EJECUTOR

- *Devuelve:* un puntero a un objeto de tipo IThread (UICL). Este objeto es el thread en el cual ejecuta el método EJECUTOR de la clase.
- *Parámetros*

semáforo:
- *E/S:* salida
- *Tipo de Datos:* semáforo de eventos (HEV)
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* representa el valor del manejador del semáforo de eventos usado por la clase TAREA para planificar la ejecución del *Intérprete de Comandos*.

11 DETENER_EJECUTOR
- *Tipo de Acceso:* público virtual
- *Parte de:* COMANDO
- *Descripción:* pone en falso el atributo booleano *CONTINUAR* para detener la ejecución del *Intérprete de Comandos*.
- *Devuelve:* no devuelve ningún valor.
- *Parámetros: no tiene.*

12. SACAR_COMANDO_COLA:
- *Tipo de Acceso:* público

- *Parte de:* COMANDO
- *Descripción:* si hay elementos en la cola de comandos ingresados por el operador, saca el primer elemento de la cola y devuelve sus valores en los parámetros de salida.
- *Devuelve:*
- *OK* si la operación terminó satisfactoriamente
- *NO_HAY_ELEMENTOS* si la cola de comandos esta vacía
- *Parámetros:*

cadena:
- *E/S:* salida
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver CMD_PARMS en *Atributos*.

p_algo:
- *E/S:* salida
- *Tipo de Datos:* puntero a VOID
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* ver P_ALGO en *Atributos*.

13. EXISTE_COMANDO:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* recorre secuencialmente el arreglo buscando un elemento cuyo nombre de comando coincida con el argumento de entrada.
- *Devuelve: true* si encuentra un elemento cuyo nombre de comando coincida con el parámetro de entrada, *false* si no lo encuentra.
- *Parámetros:*

comando: ver *BUSCAR_FUNCION*

14. VALIDAR_PARAMETROS:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* chequea

1. Que la cantidad de parámetros pasados sea la correcta
2. Que el tipo de datos de cada uno de los parámetros pasados sea el correcto

- *Devuelve:*
- *OK* si el chequeo da que está todo bien
- *ERROR_CANTIDAD_ARGUMENTOS* si la cantidad de argumentos no es correcta
- *ERROR_TIPO_ARGUMENTOS* si el tipo de algún argumento no es correcto
- *Parámetros:*

comando: ver *BUSCAR_FUNCION*

parámetros:
- *E/S:* entrada
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos.
- *Descripción:* puntero a cadena de caracteres que contiene los parámetros pasados al comando.

15. QUE_INDICE:
- *Tipo de Acceso:* público
- *Parte de:* COMANDO
- *Descripción:* recorre secuencialmente el arreglo buscando un elemento cuyo nombre de comando coincida con el argumento de entrada.

Devuelve: si encuentra un elemento cuyo nombre de comando es igual al parámetro de entrada devuelve la posición del mismo dentro del arreglo, sino devuelve -1.

- *Parámetros:*

comando: ver *BUSCAR_FUNCION*

16 ENTRARCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* COMANDO
- *Descripción:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Devuelve:* ver *ENTRARCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

17 SALIRCRITSEC:
- *Tipo de Acceso:* protegido virtual
- *Parte de:* COMANDO
- *Descripción:* ver *SALIRCRITSEC* en *tableaux_int*
- *Devuelve:* ver *SALIRCRITSEC* en *tableaux_int*
- *Parámetros*: no tiene.

2.4.2.15 Otras declaraciones que no están dentro de las Clases

1. ERROR:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -1
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa una condición de error.

2. OK:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 0
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa una condición satisfactoria.

3. DI:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 1
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el estado de alarma digital de un punto de la imagen.

4. BA:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 2
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el estado de alarma por baja de un punto de la imagen.

5. AL:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 3
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el estado de alarma por alta de un punto de la imagen.

6. TA:

- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 4
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el estado de alarma por tasa de un punto de la imagen.

7. DK:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 5
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el estado de alarma digital reconocido de un punto de la imagen.

8. BK:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 6
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el estado de alarma por baja reconocido de un punto de la imagen.

9. AK:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 7
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el estado de alarma por alta reconocido de un punto de la imagen.

10. TK:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 8
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el estado de alarma por tasa reconocido de un punto de la imagen.

11. ENTERO:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 0
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el segmento entero de la imagen.

12. REAL:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 1
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el segmento analógico de la imagen.

13. DIGITAL:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 2
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el segmento digital de la imagen.

14. REG_PARAM:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* público
- *Rango:* no aplica
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa la estructura de el registro de parámetros que recibe la rutina de atención de alarma del usuario.

14.1. P_VENTANA:
- *Tipo de Datos:* puntero a tipo IWindow (UICL)
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_PARAM
- *Descripción:* ver PVENTANA de ALARMA.

14.2. TIPO_DE_TABLEAUX:
- *Tipo de Datos:* short
- *Tipo de Acceso:* público
- *Rango:* 0..2
  0 : ENTERO
  1: REAL
  2: DIGITAL
- *Parte de:* REG_PARAM
- *Descripción:* representa el segmento de la imagen donde se encuentra el punto en estado de alarma.

14.3. P_TABLA:
- *Tipo de Datos:* puntero a TABLEAUX
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_PARAM
- *Descripción:* almacena un puntero al segmento de la imagen donde se encuentra el punto en estado de alarma.

14.4 INDICE:
- *Tipo de Datos:* entero
- *Tipo de Acceso:* público
- *Rango:* cualquier valor positivo representable por el tipo de datos
- *Parte de:* REG_PARAM
- *Descripción:* almacena el índice dentro del segmento de la imagen donde se encuentra el punto en estado de alarma.

14.5 VALOR:
- *Tipo de Datos:* double
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* REG_PARAM
- *Descripción:* almacena el valor del punto en estado de alarma.

14.6 ESTADO:
- *Tipo de Datos:* short
- *Tipo de Acceso:* público
- *Rango:* ver ESTADO en TABLEAUX_INT
- *Parte de:* REG_PARAM
- *Descripción:* almacena el estado del punto en alarma.

14.7 TIMESTAMP:
- *Tipo de Datos:* caracter de 20 posiciones
- *Tipo de Acceso:* público
- *Rango:* cualquier fecha representable en el sistema con el siguiente formato: aaaa-mm-dd.hh:mm:ss

donde:

aaaa = año
mm = mes
dd = día
hh = hora
mm = minutos
ss = segundos

- *Parte de:* REG_PARAM
- *Descripción:* representa la fecha y hora en la cual fue modificado por última vez el punto en estado de alarma.

15 BAJA:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 0
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el criterio de evaluación de alarma por baja.

16 ALTA:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 1
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el criterio de evaluación de alarma por alta.

17 TASA:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 2
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa el criterio de evaluación de alarma por tasa de cambios.

18 ABRIR_PIPE:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 0
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* este valor es usado como parámetro *mp1* en el mensaje que envía el objeto *MIMICO* de IGNATIUS al supervisor para avisarle que abra el pipe de comunicaciones entre ambos.

19 MOSTRAR_PUNTO:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 1
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* este valor es usado como parámetro *mp1* en el mensaje que envía el objeto *MIMICO* de IGNATIUS al supervisor para avisarle que debe representar en la pantalla el punto de la imagen pasado como mensaje en el pipe.

20 CERRAR_PIPE:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* 3
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* este valor es usado como parámetro *mp1* en el mensaje que envía el objeto *MIMICO* de IGNATIUS al supervisor para avisarle que cierre el pipe de comunicaciones entre ambos.

21 STRUCT_COLA:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* público
- *Rango:* no aplica
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa la estructura de información usada por DosDevIOCtl para hacer una consulta a las
colas de transmisión y recepción [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

21.1 NUMCAR:
- *Tipo de Datos:* USHORT
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* STRUCT_COLA
- *Descripción:* representa la cantidad de caracteres que hay en la cola [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

21.2 TAMANIOCOLA:
- *Tipo de Datos:* USHORT
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* STRUCT_COLA
- *Descripción:* representa el tamaño de la cola física del device driver [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22 STRUCT_DCB:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* público
- *Rango:* no aplica
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa la estructura de información usada por DosDevIOCtl. en las operaciones de consulta y seteo del Device Control Block [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.1 WRITETIMEOUT:
- *Tipo de Datos:* USHORT
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* STRUCT_DCB
- *Descripción:* representa el período de tiempo, expresado en centésimas de segundo, usado para el procesamiento del timeout en la escritura [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.2 READTIMEOUT:
- *Tipo de Datos:* USHORT
- *Tipo de Acceso:* público

- *Rango:* cualquier valor representable por el tipo de datos
- *Parte de:* STRUCT_DCB
- *Descripción:* representa el período de tiempo, expresado en centésimas de segundo, usado para el procesamiento del timeout en la lectura [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.3 FLAGS1:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*

Bits 0-1 DTR Control mode.

Bit 1	Bit 0	Descripción
0	0	Disable
0	1	Enable
1	0	Input handshaking
1	1	Invalid input. General failure error.

Bit 2 Reservado (retorna 0)
Bit 3 Enable output handshaking using CTS
Bit 4 Enable output handshaking using DSR
Bit 5 Enable output handshaking using DCD
Bit 6 Enable input sensitivity using DSR
Bit 7 Reservado (retorna 0)

- *Parte de:* STRUCT_DCB
- *Descripción:* contiene información del Device Control Block [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.4 FLAGS2:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*

Bit 0 Enable Automatic Transmit Flow Control (XON/XOFF)
Bit 1 Enable Automatic Receive Flow Control (XON/XOFF)
Bit 2 Enable error replacement character
Bit 3 Enable null stripping (remove null bytes)
Bit 4 Enable break replacement character
Bit 5 Automatic Receive Flow Control
0 = Normal
1 = Full-Duplex

Bits 6-7 RTS Control mode.

Bit 7	Bit 6	Descripción
0	0	Disable
0	1	Enable
1	0	Input handshaking
1	1	Toggling on transmit

- *Parte de:* STRUCT_DCB
- *Descripción:* contiene información del Device Control Block [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.5 FLAGS3:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*

Bit 0 Enable Write Infinite Timeout processing
Bits 1-2 Read Timeout processing.

Bit 2	Bit 1	Descripción

0	1	Normal Read Timeout processing
1	0	Wait-For-Something, Read Timeout processing
1	1	No-Wait, Read Timeout processing

Bits 3-4 Extended Hardware Buffering.

Bit 4	Bit 3	Descripción
0	0	Not supported
0	1	Extended Hardware Buffering Disabled
1	0	Extended Hardware Buffering Enabled
1	1	Automatic Protocol Override

Bits 5-6 Receive Trigger Level.

Bit 6	Bit 5	Descripción
0	0	1 caracter
0	1	4 caracteres
1	0	8 caracteres
1	1	14 caracteres

Bit 7 Transmit Buffer Load Count
0 = 1 caracter
1 = 16 caracteres

- *Parte de:* STRUCT_DCB
- *Descripción:* contiene información del Device Control Block [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.6 ERRORCHAR:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango: 00h..FFh*
- *Parte de:* STRUCT_DCB
- *Descripción:* caracter de reemplazo por error [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.7 BREAKCHAR:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango: 00h..FFh*
- *Parte de:* STRUCT_DCB
- *Descripción:* caracter de reemplazo por condiciones de break [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.8 XONCHAR:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango: 00h..FFh*
- *Parte de:* STRUCT_DCB
- *Descripción:* caracter de control de flojo de transmisión XON [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

22.9 XOFFCHAR:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango: 00h..FFh*
- *Parte de:* STRUCT_DCB
- *Descripción:* caracter de control de flojo de transmisión XOFF [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

23 STRUCT_LCB:

- *Tipo de Datos:* estructura
- *Tipo de Acceso:* público
- *Rango:* no aplica
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa la estructura de datos para consultar las características de la línea [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

23.1 DATABITS:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*
  0x00-0x04
    Reservado
  0x05
    5 bits de datos
  0x06
    6 bits de datos
  0x07
    7 bits de datos (valor inicial)
  0x08
    8 bits de datos
  0x09-0xFF
    Reservado
- *Parte de:* STRUCT_LCB
- *Descripción:* bandera de bits de datos [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

23.2 PARITY:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*
  0x00
    Sin paridad
  0x01
    Paridad par
  0x02
    Paridad impar (valor inicial)
  0x03
    Paridad marca (bit de paridad siempre 1)
  0x04
    Parity espacio (bit de paridad siempre 0)
  0x05-0xFF
    Reservado
- *Parte de:* STRUCT_LCB
- *Descripción:* bandera de bits de paridad [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

23.3 STOPBITS:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*
  0x00
    1 bit de stop (valor inicial)
  0x01
    1.5 bits de stop (válido únicamente con longitud de 5-bit WORD)

0x02
2 bits de stop (no válido con longitud de 5-bit WORD)
0x03-0xFF
Reservado

- *Parte de:* STRUCT_LCB
- *Descripción:* bandera de bits de stop [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

23.4 TRXBREAK:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*
  0 : Sin break de transmisión
  1 : Con break de transmisión
- *Parte de:* STRUCT_LCB
- *Descripción:* bandera de break de transmisión [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

24 STRUCT_LCB2:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* público
- *Rango:* no aplica
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa la estructura de datos para setear las características de la línea [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

24.1 DATABITS:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*
  00h-04h Reserved
  05h 5 bits de datos
  06h 6 bits de datos
  07h 7 bits de datos (valor inicial)
  08h 8 bits de datos
  09h-FFh Reservado
- *Parte de:* STRUCT_LCB2
- *Descripción:* bandera de bits de datos [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

24.2 PARITY:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*
  00h Sin paridad
  01h Paridad par
  02h Paridad impar (valor inicial)
  03h Paridad marca (bit de paridad siempre 1)
  04h Paridad espacio (bit de paridad siempre 0)
  05h-FFh Reservado
- *Parte de:* STRUCT_LCB2
- *Descripción:* bandera de bits de paridad [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

24.3 STOPBITS:

- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:*
  00h 1 bit de stop (valor inicial)
  01h 1.5 bits de stop (válido únicamente con longitud de 5-bit WORD)
  02h 2 bits de stop (no válido con longitud de 5-bit WORD)
  03h-FFh Reservado
- *Parte de:* STRUCT_LCB2
- *Descripción:* bandera de bits de stop [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

25. STRUCT_MCB:
- *Tipo de Datos:* estructura
- *Tipo de Acceso:* público
- *Rango:* no aplica
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa la estructura de datos para setear las señales de control del módem [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

25.1 BFMASKON:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:* [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference]
- *Parte de:* STRUCT_MCB
- *Descripción:* bandera de bits para setear DTR y RTS [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

25.2 BFMASKOFF:
- *Tipo de Datos:* BYTE
- *Tipo de Acceso:* público
- *Rango:* [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference]
- *Parte de:* STRUCT_MCB
- *Descripción:* bandera de bits para setear DTR y RTS [ver IOCtl Commands for RS232 en OS/2 Control Program Guide and Reference].

26 COMANDO_NO_VALIDO:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -9
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que un comando no es válido.

27 NO_HAY_ELEMENTOS:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -8
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que no hasy más elementos.

28 MASCARA_INCORRECTA:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -7

- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que la máscara de edición de parámetros es incorrecta.

29 ERROR_TIPO_ARGUMENTOS:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -6
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que hay un error en el tipo de argumentos.

30 ERROR_CANTIDAD_ARGUMENTOS:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -5
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que hay un error en la cantidad de argumentos.

31 INDICE_INCORRECTO:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -4
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que el índice provisto es incorrecto.

32 NO_HAY_ESPACIO:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -3
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que no hay mas espacio para agregar en el arreglo.

33 COMANDO_YA_EXISTE:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -2
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que el comando que se quiere agregar a la tabla de comandos existía previamente.

34 COMANDO_INEXISTENTE:
- *Tipo de Datos:* constante
- *Tipo de Acceso:* público
- *Valor:* -1
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* esta constante es usada por la clase *COMANDO* para informar que el comando provisto no existe en la tabla de comandos.

35 P_OBJETOS:
- *Tipo de Datos:* estructura

- *Tipo de Acceso:* público
- *Rango:* no aplica
- *Parte de:* archivo header IGNATIUS.HPP
- *Descripción:* representa la estructura de punteros a todos los objetos de IGNATIUS.

35.1 P_TABLEAUX:
- *Tipo de Datos:* puntero a TABLEAUX
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo TABLEAUX.

35.2 P_ALARMA:
- *Tipo de Datos:* puntero a ALARMA
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo ALARMA.

35.3 P_COLA_IN:
- *Tipo de Datos:* puntero a COLA
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo COLA que representa la cola de entrada.

35.4 P_COLA_OUT:
- *Tipo de Datos:* puntero a COLA
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo COLA que representa la cola de salida

35.5 P_HISTORICO:
- *Tipo de Datos:* puntero a HISTORICO
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo HISTORICO.

35.6 P_TAREA:
- *Tipo de Datos:* puntero a TAREA
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo TAREA.

35.7 P_PROCESO:
- *Tipo de Datos:* puntero a PROCESO
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo PROCESO.

35.8 P_MSG:

- *Tipo de Datos:* puntero a MSG
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo MSG.

35.9 P_PIPE:
- *Tipo de Datos:* puntero a PIPE
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo PIPE.

35.10 P_MIMICO:
- *Tipo de Datos:* puntero a MIMICO
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo MIMICO.

35.11 P_PORT:
- *Tipo de Datos:* puntero a PORT
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* P_OBJETOS
- *Descripción:* puntero al objeto de tipo PORT.

36. FUNCION:
- *Tipo de Datos:* definición de tipo
- *Tipo de Acceso:* público
- *Parte de:* header file IGNATIUS.HPP
- *Descripción:* prototipo de función de atención de comando. La clase COMANDO asocia una función de tipo *FUNCION* para cada uno de los comandos del arreglo de comandos. Esta función encapsula el comportamiento del comando asociado.
- *Devuelve:* cualquier valor entero definido por la rutina de atención de comandos.
- *Parámetros:*

char *:
- *E/S:* entrada
- *Tipo de Datos:* puntero a caracter.
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* la cadena de caracteres apuntada por este parámetro es la que contiene los valores ingresados como parámetros al comando.

p_objetos *:
- *E/S:* entrada
- *Tipo de Datos:* puntero a P_OBJETOS.
- *Rango:* cualquier valor representable por el tipo de datos
- *Descripción:* representa el puntero a la estructura que contiene los punteros a todos los objetos de IGNATIUS. Esto le permite a la rutina de atención de comandos tener acceso a todos los objetos de IGNATIUS.

void *:
- *E/S:* entrada
- *Tipo de Datos:* puntero a VOID.
- *Rango:* cualquier valor representable por el tipo de datos

- *Descripción:* este parámetro no tiene una función específica. Puede ser utilizado para cualquier fin de utilidad para la rutina de atención de comandos ya que se puede pasar un puntero a cualquier objeto.

37 PFUNCION:
- *Tipo de Datos:* definición de tipo puntero a FUNCION
- *Tipo de Acceso:* público
- *Rango:* cualquier valor representable por el tipo de datos.
- *Parte de:* header file IGNATIUS.HPP
- *Descripción:* puntero a función de tipo FUNCION.

38. TIMESTAMP:
- *Tipo de Acceso:* público externo
- *Parte de:* header file IGNATIUS.HPP
- *Descripción:* rutina que devuelve un puntero a una cadena de caracteres que contiene la fecha y hora actual.
- *Devuelve:* un puntero a una cadena de caracteres con la siguiente forma: aaaa-mm-dd.hh:mm:ss
donde:
aaaa = año
mm = mes
dd = día
hh = hora
mm = minutos
ss = segundos
- *Parámetros:* no tiene.

2.5 Usando la Biblioteca

Las clases se encuentran almacenadas en una biblioteca C++ cuyo nombre de archivo es IGNATIUS.LIB. El archivo header se llama IGNATIUS.HPP. Los programas que usen esta biblioteca deben ser escritos usando el compilador VisualAge C++ 3.0 para OS/2.

A modo de ejemplo de uso de la biblioteca se provee el código fuente de un Supervisor (Ver Apéndice XXX).

Para compilar y linkeditar su programa usando la biblioteca siga los siguientes pasos:

1. Incluya dentro de la declaración de headers de su programa el archivo header de la biblioteca IGNATIUS.HPP:

```
#include "IGNATIUS.HPP"
```

2. Compile su programa usando VisualAge C++ 3.0 para OS/2.

3. Linkedite el módulo objeto de su programa con VisualAge C++ 3.0 para OS/2 incluyendo como biblioteca adicional a IGNATIUS.LIB.

A continuación se muestra a modo de ejemplo la macro para hacer MAKE del Supervisor provisto como ejemplo:

```
# Makefile
#
#       variables para las opciones de Build
#
CFLAGS=/c /Ti /Gm /Gd /W3 /Tm
```

```
LFLAGS=/ST:65536 /ALIGN:4 /DE /NOI /NOE
L_ADDITIONAL_LIBS=IGNATIUS.lib eppom30.lib eppooc3.lib
L_ADDITIONAL_OBJS=
#
#       Definiciones adicionales
#
CFLAGS=/Ti- $(CFLAGS) /I . /I ..\SUBPROCS
L_ADDITIONAL_LIBS=$(L_ADDITIONAL_LIBS) VSBGRAPH.LIB
SUBPROCS=..\\SUBPROCS\\
EXCL=!
OBJECTS= $(L_ADDITIONAL_OBJS) \
        MAIN.OBJ \
        VPALARMA.OBJ \
        VPALARM1.OBJ \
        VPHISTOR.OBJ \
        VPIMAGEN.OBJ \
        VPMIMICO.OBJ \
        VPPORT.OBJ \
        VPPORTIM.OBJ \
        VPPROCES.OBJ \
        VPTENDEN.OBJ \
        PROC_USU.OBJ \
        COMPORT.OBJ \
        USERCMD.OBJ \
        USERHDR.OBJ

ALL: RUN.EXE
#
#       Sentencias para compilar las clases
#
MAIN.OBJ:  MAIN.CPP MAIN.HPP MAIN.RCH
   ICC $(CFLAGS) MAIN.CPP
VPALARMA.OBJ:  VPALARMA.CPP VPALARMA.HPP VPALARMA.RCH
   ICC $(CFLAGS) VPALARMA.CPP
VPALARM1.OBJ:  VPALARM1.CPP VPALARM1.HPP VPALARM1.RCH
   ICC $(CFLAGS) VPALARM1.CPP
VPHISTOR.OBJ:  VPHISTOR.CPP VPHISTOR.HPP VPHISTOR.RCH
   ICC $(CFLAGS) VPHISTOR.CPP
VPIMAGEN.OBJ:  VPIMAGEN.CPP VPIMAGEN.HPP VPIMAGEN.RCH
   ICC $(CFLAGS) VPIMAGEN.CPP
VPMIMICO.OBJ:  VPMIMICO.CPP VPMIMICO.HPP VPMIMICO.RCH
   ICC $(CFLAGS) VPMIMICO.CPP
VPPORT.OBJ:  VPPORT.CPP VPPORT.HPP VPPORT.RCH
   ICC $(CFLAGS) VPPORT.CPP
VPPORTIM.OBJ:  VPPORTIM.CPP VPPORTIM.HPP VPPORTIM.RCH
   ICC $(CFLAGS) VPPORTIM.CPP
VPPROCES.OBJ:  VPPROCES.CPP VPPROCES.HPP VPPROCES.RCH
   ICC $(CFLAGS) VPPROCES.CPP
VPTENDEN.OBJ:  VPTENDEN.CPP VPTENDEN.HPP VPTENDEN.RCH
   ICC $(CFLAGS) VPTENDEN.CPP
PROC_USU.OBJ:  $(SUBPROCS)PROC_USU.CPP
   ICC $(CFLAGS) $(SUBPROCS)PROC_USU.CPP
COMPORT.OBJ:  $(SUBPROCS)COMPORT.CPP
   ICC $(CFLAGS) $(SUBPROCS)COMPORT.CPP
USERCMD.OBJ:  $(SUBPROCS)USERCMD.CPP
   ICC $(CFLAGS) $(SUBPROCS)USERCMD.CPP
USERHDR.OBJ:  $(SUBPROCS)USERHDR.CPP
   ICC $(CFLAGS) $(SUBPROCS)USERHDR.CPP
#
#       Sentencias para compilar los recursos de las clases
#
PROJECT$(EXCL).RES:  PROJECT$(EXCL).RC \
          MAIN.RCH MAIN.RC \
          VPALARMA.RCH VPALARMA.RC \
          VPALARM1.RCH VPALARM1.RC \
          VPHISTOR.RCH VPHISTOR.RC \
          VPIMAGEN.RCH VPIMAGEN.RC \
          VPMIMICO.RCH VPMIMICO.RC \
          VPPORT.RCH VPPORT.RC \
          VPPORTIM.RCH VPPORTIM.RC \
          VPPROCES.RCH VPPROCES.RC \
          VPTENDEN.RCH VPTENDEN.RC
```

```
  RC -r PROJECT$(EXCL).RC
#
#     Sentencias para linkeditar en ejecutable
#
RUN.EXE:  $(OBJECTS) PROJECT$(EXCL).DEF PROJECT$(EXCL).RES PROJECT$(EXCL).MAK
  ICC @<<
        /Tdp /B"/PM:PM $(LFLAGS) "
        /Fe"RUN.EXE"
        $(OBJECTS)
        $(L_ADDITIONAL_LIBS)
        PROJECT!.DEF
<<
  RC PROJECT$(EXCL).RES RUN.EXE
```

2.6 *Ejemplo de un supevisor que usa IGNATIUS.*

Cada una de las clases de la biblioteca IGNATIUS fue sometida a las siguientes pruebas:

1. *Prueba Unitaria:* el objetivo de esta prueba fue testear todas y cada una de las funciones de la clase a probar independientemente del funcionamiento integrado con el resto de las clases de la biblioteca. Se generó un programa de prueba a tal efecto para cada una de las clases.

2. *Prueba de Integración:* el objetivo de esta prueba fue testear a cada una de las clases en cuanto a su funcionamiento integrado con el resto de las clases de la biblioteca. A los programas desarrollados para las pruebas unitarias se les agregó código para efectuar esta prueba. Se puso foco en probar las interfaces y la comunicación entre las clases.

3. *Prueba del Sistema.* El objetivo de esta prueba es garantizar el correcto funcionamiento y la integridad total del sistema. A tal efecto se desarrolló una aplicación SCADA usando IGNATIUS.

El proceso de prueba se realizó de forma cíclica hasta eliminar la totalidad de los errores detectados. Por cada error detectado en cada una de las pruebas, luego de su corrección, se realizaron nuevamente todos las pruebas mencionadas. Por ejemplo, si en la Prueba del Sistema se detectó un error en la clase de Alarmas, se corrigió el error, se realizó la Prueba Unitaria de la clase de Alarmas, se realizó la Prueba de Integración de la clase de Alarmas con el resto de las clases y finalmente se ejecutó la Prueba del Sistema.

Como se menciona antes, para realizar una prueba total de la herramienta se construyó un supervisor. Este supervisor, puede ser usado como ejemplo para la construcción de otros supervisores. El código fuente del mismo se encuentra listado en el Apéndice C. A continuación se describe como se implementaron los niveles 1 y 2 del SACADA ejemplo usando IGNATIUS.

2.6.1 Servicios de Interfaz (Nivel 1)

Para la codificación de los *servicios de interfaz* se utilizó una herramienta de programación visual que agilizó notablemente la construcción de las ventanas y todos los objetos gráficos. A continuación se describen los objetos que componen los Servicios de Interfaz:

2.6.1.1 Main

El objeto *Main* es el que provee la ventana principal del supervisor. Las tareas llevadas a cabo por el mismo son:

1. Creación de los objetos de IGNATIUS (imagen, alarma, histórico, etc.)
2. Inicialización de variables
3. Definición de las rutinas de atención de comandos
4. Creación de la ventana principal del supervisor y de todos los objetos que la componen (botones, barra de acción, campo de ingreso de datos, etc.)
5. Creación del pipe de comunicaciones.
6. Creación de un prompt para el ingreso de comandos
7. Planificación de las Tareas
8. Lanzamiento del motor de ejecución.
9. Detención del motor de ejecución.
10. Invocación de las restantes ventanas de la aplicación según selección del usuario.
11. Redefinición del manejador del evento de movimiento del mouse para mostrar la posición del mismo.

2.6.1.2 vpAlarma

El objeto *vpAlarma* es el que provee al operador un front-end para la actualización y monitoreo de las condiciones de alarma. Las tareas llevadas a cabo por el mismo son:

1. Inicialización de variables
2. Definición de las rutinas de atención de alarma
3. Creación de los objetos de la ventana de actualización de alarmas
4. Visualización de las alarmas existentes
5. Actualización de alarmas

2.6.1.3 vpAlarmaImagen

El objeto *vpAlarmaImagen* es el que provee al operador un front-end para la actualización y monitoreo de las relaciones entre las condiciones de alarma y los puntos de la imagen. Las tareas llevadas a cabo por el mismo son:

1. Inicialización de variables
2. Creación de los objetos de la ventana de relaciones
3. Visualización de las alarmas existentes
4. Visualización de los puntos de la imagen existentes
5. Visualización de las relaciones entre las condiciones de alarma y los puntos de la imagen
6. Actualización de las relaciones

2.6.1.4 vpHistorico

El objeto *vpHistorico* es el que provee al operador un front-end para monitorear e imprimir los registros almacenados en el archivo histórico. Las tareas llevadas a cabo por el mismo son:

1. Inicialización de variables

2. Creación de los objetos de la ventana de registros del archivo histórico
3. Visualización de los registros del archivo histórico
4. Impresión de los registros del archivo histórico

2.6.1.5 vpImagen

El objeto *vpImagen* es el que provee al operador un front-end para la actualización y monitoreo de los valores de los puntos de la imagen. Las tareas llevadas a cabo por el mismo son:

1. Inicialización de variables
2. Creación de los objetos de la ventana de actualización de imagen
3. Visualización de los puntos de la imagen existentes
4. Actualización de los puntos de la imagen

2.6.1.6 vpMimico

El objeto *vpMimico* es el que provee al operador un front-end para la actualización y monitoreo de la representación de los procesos y los valores de los puntos de la imagen asociados a los mismos (mímicos). Las tareas llevadas a cabo por el mismo son:

1. Inicialización de variables
2. Creación de los objetos de la ventana de mímicos
3. Visualización de los puntos de la imagen existentes
4. Visualización de los procesos existentes
5. Visualización de los mímicos existentes
6. Actualización de los mímicos

2.6.1.7 vpPort

El objeto *vpPort* es el que provee al operador un front-end para la declaración y monitoreo de los ports conectados a los dispositivos. Las tareas llevadas a cabo por el mismo son:

1. Inicialización de variables
2. Creación de los objetos de la ventana de ports
3. Visualización de los ports declarados
4. Actualización de las declaraciones de ports

2.6.1.8 vpPortImagen

El objeto *vpPortImagen* es el que provee al operador un front-end para la actualización y monitoreo de las relaciones entre los ports conectados a los dispositivos y los puntos de la imagen. Las tareas llevadas a cabo por el mismo son:

1. Inicialización de variables
2. Creación de los objetos de la ventana de relaciones
3. Visualización de los puntos de la imagen existentes
4. Visualización de los ports declarados
5. Visualización de las relaciones entre los ports y los puntos de la imagen
6. Actualización de las relaciones

2.6.1.9 vpProceso

El objeto *vpProceso* es el que provee al operador un front-end para la actualización y monitoreo de los procesos del mundo exterior. Las tareas llevadas a cabo por el mismo son:

1. Inicialización de variables
2. Creación de los objetos de la ventana de actualización de procesos
3. Visualización de los procesos existentes
4. Actualización de los procesos
5. Monitoreo en la ventana *Main* del proceso seleccionado para monitorear
6. Selección visual de la imagen asociada al proceso

2.6.2 Servicios Particulares (Nivel 2)

Los *servicios particulares* fueron implementados como programas C++. Los tres programas que componen estos servicios son los siguientes:

2.6.2.1 USERHDR.CPP

Este programa es el manejador de mensajes del usuario. Su función principal es capturar los mensajes enviados a la ventana Main por los objetos *MIMICO*, *vpProceso* y por las rutinas de atención de comandos. Los mensajes procesados son los siguientes:

1. ABRIR_PIPE: mensaje enviado por el objeto *MIMICO*. Abre el pipe de comunicaciones entre el objeto *MIMICO* y el supervisor

2. MOSTRAR_PUNTO: mensaje enviado por el objeto *MIMICO*. Lee un mensaje del pipe de comunicaciones y lo representa por pantalla

3. CERRAR_PIPE: mensaje enviado por el objeto *MIMICO*. Cierra el pipe de comunicaciones entre el objeto *MIMICO* y el supervisor

4. MOSTRAR_IMAGEN: mensaje enviado por las rutinas de atención de comandos. Crea la ventana de actualización de la imagen.

5. MOSTRR_PROCESOS: mensaje enviado por las rutinas de atención de comandos. Crea la ventana de actualización de procesos.

6. MOSTRAR_ALARMAS: mensaje enviado por las rutinas de atención de comandos. Crea la ventana de actualización de condiciones de alarma.

7. MOSTRAR_PORTS: mensaje enviado por las rutinas de atención de comandos. Crea la ventana de actualización de ports.

8. MOSTRAR_HISTORICO: mensaje enviado por las rutinas de atención de comandos. Crea la ventana de monitoreo de registros históricos.

9. EJECUTAR: mensaje enviado por las rutinas de atención de comandos. Pone en funcionamiento el motor de ejecución.

10. DETENER: mensaje enviado por las rutinas de atención de comandos. Detiene el funcionamiento del motor de ejecución.

11. SALIR: mensaje enviado por las rutinas de atención de comandos. Termina la ejecución del Supervisor.

12. BORRAR_COLA: mensaje enviado por el objeto *vpProceso*. Borra la cola de elementos de representación del proceso en monitor.

13. MOSTRAR_BITMAP: mensaje enviado por el objeto *vpProceso*. Muestra en la ventana *Main* el bitmap asociado al proceso en monitor.

14. MOSTRAR_SEMAFORO: mensaje enviado por las rutinas de atención de comandos. Muestra en la ventana *Main* el bitmap asociado al semáforo de estados de alarma.

Cualquier otro mensaje que no esté contenido dentro de los mensajes mencionados anteriormente es pasado al manejador de mensajes del sistema operativo.

2.6.2.2 PROC_USU.CPP

Este programa contiene las rutinas de atención de alarmas codificadas por el usuario. Estas rutinas deben ser funciones de la forma:

void _System NombreRutina(ULONG param)

El parámetro *param* debe ser tratado como un puntero a una estructura *reg_param* (ver *REG_PARAM* en el capítulo Otras declaraciones que no están dentro de las Clases)

Estas rutinas son ejecutadas en un thread independiente por el Analizador de Alarmas. A modo de ejemplo se codificaron cinco rutinas a saber:

1. PROC_USU1: realiza las siguientes tareas:
   1. 1 Envía un mensaje a la *Main* para decirle que muestre el bitmap de semáforo en rojo.
   1.2 Crea una ventana con los valores del punto en alarma y pregunta si el operador quiere reconocer la alarma.
   1.3 Si el operador reconoce la alarma, entonces le envía un mensaje a la *Main* para decirle que muestre el bitmap de semáforo en verde.
   1.4 Termina la ejecución del thread.

2. PROC_USU2: realiza las siguientes tareas:
   2.1 Crea una ventana con el mensaje de que se ejecuto el procedimiento de atención de alarma PROC_USU2.
   2.2 Termina la ejecución del thread.

3. PROC_USU3: realiza las siguientes tareas:
   3.1 Crea una ventana con el mensaje de que se ejecuto el procedimiento de atención de alarma PROC_USU3.
   3.2 Termina la ejecución del thread.

4. PROC_USU4: realiza las siguientes tareas:
   4.1 Crea una ventana con el mensaje de que se ejecuto el procedimiento de atención de alarma PROC_USU4.

4.2 Termina la ejecución del thread.

5. PROC_USU5: realiza las siguientes tareas:
   5.1 Crea una ventana con el mensaje de que se ejecuto el procedimiento de atención de alarma PROC_USU5.
   5.2 Termina la ejecución del thread.

2.6.2.3 USERCMD.CPP

Este programa contiene las rutinas de atención de comandos codificadas por el usuario. Estas rutinas deben ser funciones del tipo *FUNCION* (ver *FUNCION* en el capítulo Otras declaraciones que no están dentro de las Clases). Las siguientes rutinas fueron codificadas a modo de ejemplo:

1. MONITOREAR_PROCESO: realiza las siguientes tareas:
   1.1 Inicializar variables
   1.2 Si el id de proceso pasado como parámetro no es válido, entonces:
      1.2.1 Mostrar error por pantalla
   1.3 Sino:
      1.3.1 Marcar como proceso en monitor el proceso especificado
      1.3.2 Buscar el bitmap asociado al proceso
      1.3.3 Enviar el mensaje BORRAR_COLA a la ventana *Main*
      1.3.4 Enviar el mensaje MOSTRAR_BITMAP a la ventana *Main*
      1.3.5 devolver OK

2. MOSTRAR_IMAGEN: realiza las siguientes tareas:
   2.1 Inicializar variables
   2.2 Enviar el mensaje MOSTRAR_IMAGEN a la ventana *Main*
   2.3 devolver OK

3. MOSTRAR_PROCESOS: realiza las siguientes tareas:
   3.1 Inicializar variables
   3.2 Enviar el mensaje MOSTRAR_PROCESOS a la ventana *Main*
   3.3 devolver OK

4. MOSTRAR_ALARMAS: realiza las siguientes tareas:
   4.1 Inicializar variables
   4.2 Enviar el mensaje MOSTRAR_ALARMAS a la ventana *Main*
   4.3 devolver OK

5. MOSTRAR_PORTS: realiza las siguientes tareas:
   5.1 Inicializar variables
   5.2 Enviar el mensaje MOSTRAR_PORTS a la ventana *Main*
   5.3 devolver OK

6. MOSTRAR_HISTORICO: realiza las siguientes tareas:
   6.1 Inicializar variables
   6.2 Enviar el mensaje MOSTRAR_HISTORICO a la ventana *Main*
   6.3 devolver OK

7. EJECUTAR: realiza las siguientes tareas:
   7.1 Inicializar variables
   7.2 Enviar el mensaje EJECUTAR
   7.3 devolver OK

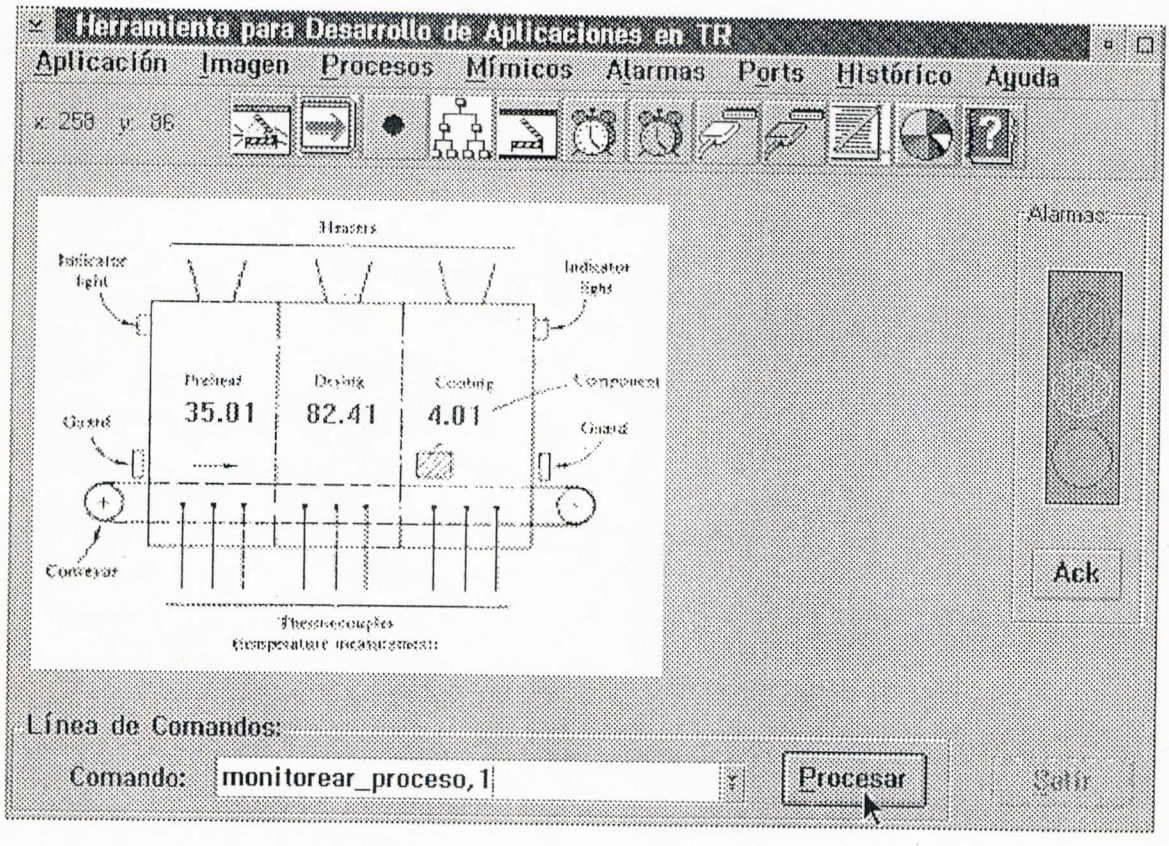
8. DETENER: realiza las siguientes tareas:
   8.1 Inicializar variables
   8.2 Enviar el mensaje DETENER
   8.3 devolver OK

9. SALIR: realiza las siguientes tareas:
   9.1 Inicializar variables
   9.2 Enviar el mensaje SALIR
   9.3 devolver OK

3. Apéndice A - Ejemplo 1 de SCADA que usa IGNATIUS

3.1 Ventana Principal

La ventana principal es la primera en aparecer cuando se ejecuta el supervisor. Desde ella se pueden invocar el resto de las ventanas de la aplicación.



La ventana está compuesta por 5 secciones principales:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana. Estas acciones permiten acceder a las otras ventanas de la aplicación, arrancar el motor de ejecución, detener el motor de ejecución, visualizar el archivo histórico, salir de la aplicación, etc..

2. **Barra de Iconos:** contiene iconos que representan gráficamente acciones que también se pueden acceder a través de la Barra de Acciones. Es una manera más amigable que la barra de acciones de indicar un comportamiento a la aplicación.

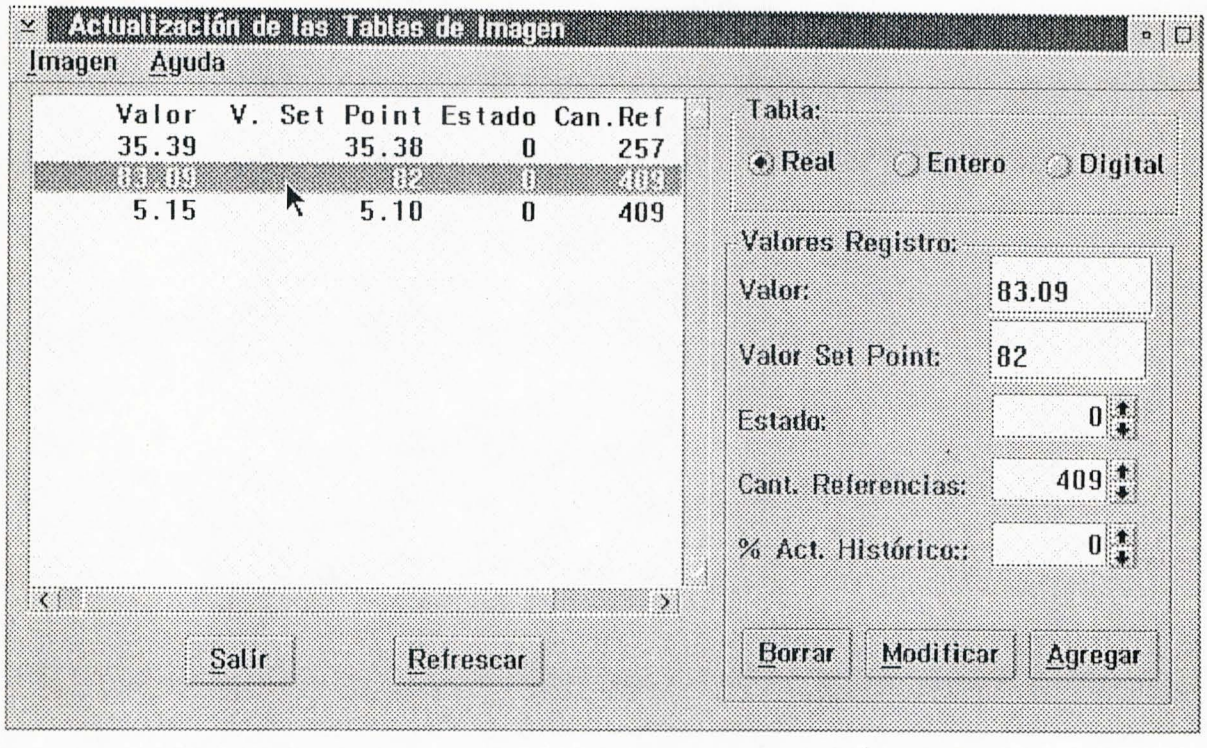
3. **Bitmap de Proceso:** en esta sección se despliega el bitmap del proceso que se está monitoreando.

4. **Semáforo de Alarma:** este bitmap permite identificar de una manera gráfica los estados de alarma. Cuando el semáforo se encuentra en rojo indica que hay algún punto de la imagen en estado de alarma, cuando está en verde indica que no hay ningún punto en estado de alarma.

5. **Prompt de Comando:** este objeto contiene una lista de los comandos válidos para la aplicación, lo que agiliza el ingreso de comandos y minimiza la introducción de errores.

3.2 Actualización de las Tablas de Imagen

En esta ventana se muestran los puntos de la imagen definidos por el operador, sus valores y se permite la actualización de los mismos. Es invocada desde la Ventana Principal.



Esta ventana se encuentra dividida en 4 secciones:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: agregar un punto de la imagen, modificarlo, borrarlo, refrescar la información, salir, ayuda.

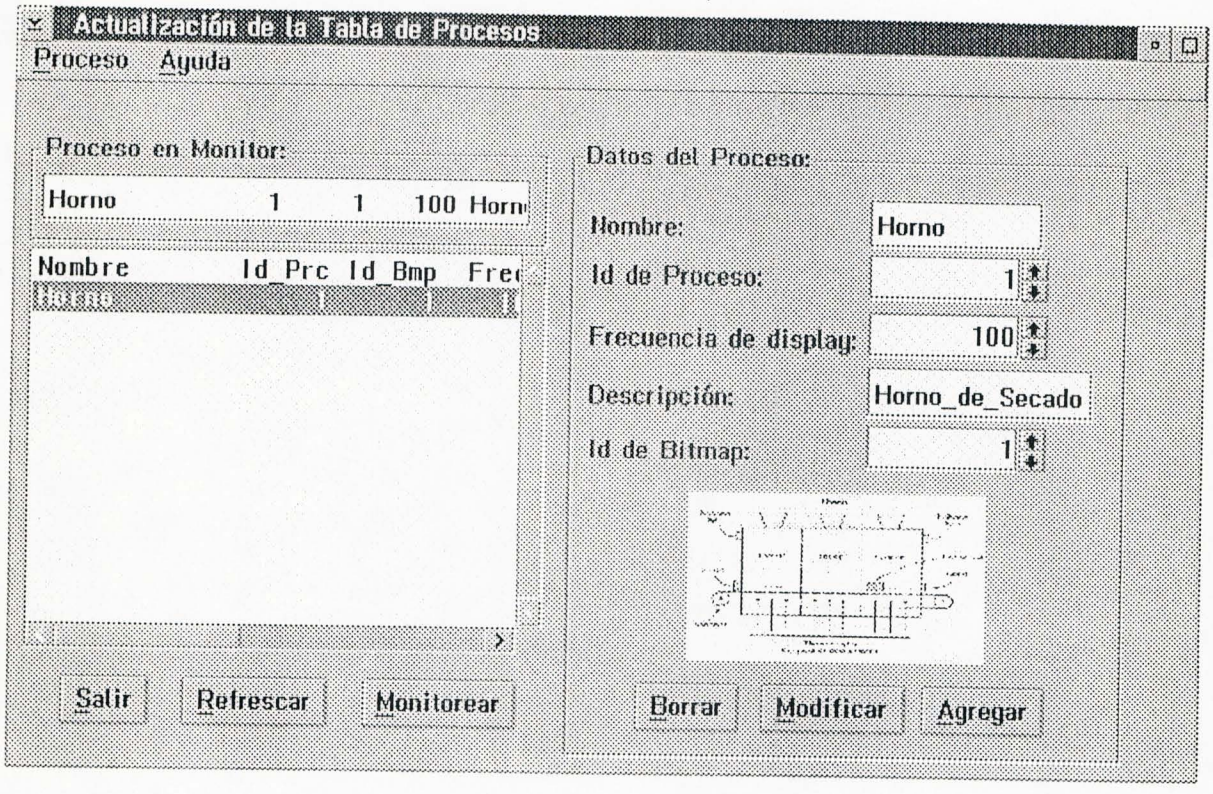
2. **Segmento Virtual de la Imagen:** en la parte derecha superior de la ventana se encuentran tres objetos de tipo RadioButton que permiten indicar el tipo de segmento de la imagen sobre el que se quiere operar: Entero, Real o Digital. Cada vez que se selecciona un segmento, la Tabla de Puntos de la Imagen refleja los valores de los puntos del segmento seleccionado, y todas las operaciones solicitadas de aquí en más se harán en ese espacio virtual.

3. **Datos del Punto de la Imagen:** esta sección está ubicada en la parte derecha inferior de la ventana. Representa todos los datos de un punto de la imagen: Valor, Valor del Set Point, Estado, Cantidad de Referencias, Porcentaje de Actualización en el Archivo Histórico. También se encuentran tres objetos de tipo Button para las acciones de *Agregar*, *Modificar* y *Borrar* un punto de la imagen.

4. **Tabla de Puntos de la Imagen:** en la parte izquierda y central de la ventana se encuentra un objeto de tipo ListBox que contiene todos los puntos de la imagen del segmento seleccionado que fueron previamente definidos por el operador del sistema. Los datos de los puntos de la imagen se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. Se pueden marcar puntos viendo reflejados sus valores en la sección de Datos del Punto de la Imagen. El objeto de tipo Button *Refrescar* permite refrescar la información de los puntos de la imagen definidos.

3.3 Actualización de la Tabla de Procesos

En esta ventana se muestran los procesos definidos por el operador, sus valores y se permite la actualización de los mismos. Es invocada desde la Ventana Principal.



Esta ventana se encuentra dividida en 4 secciones:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: agregar una definición de proceso, modificarla, borrarla, refrescar la información, salir, ayuda.

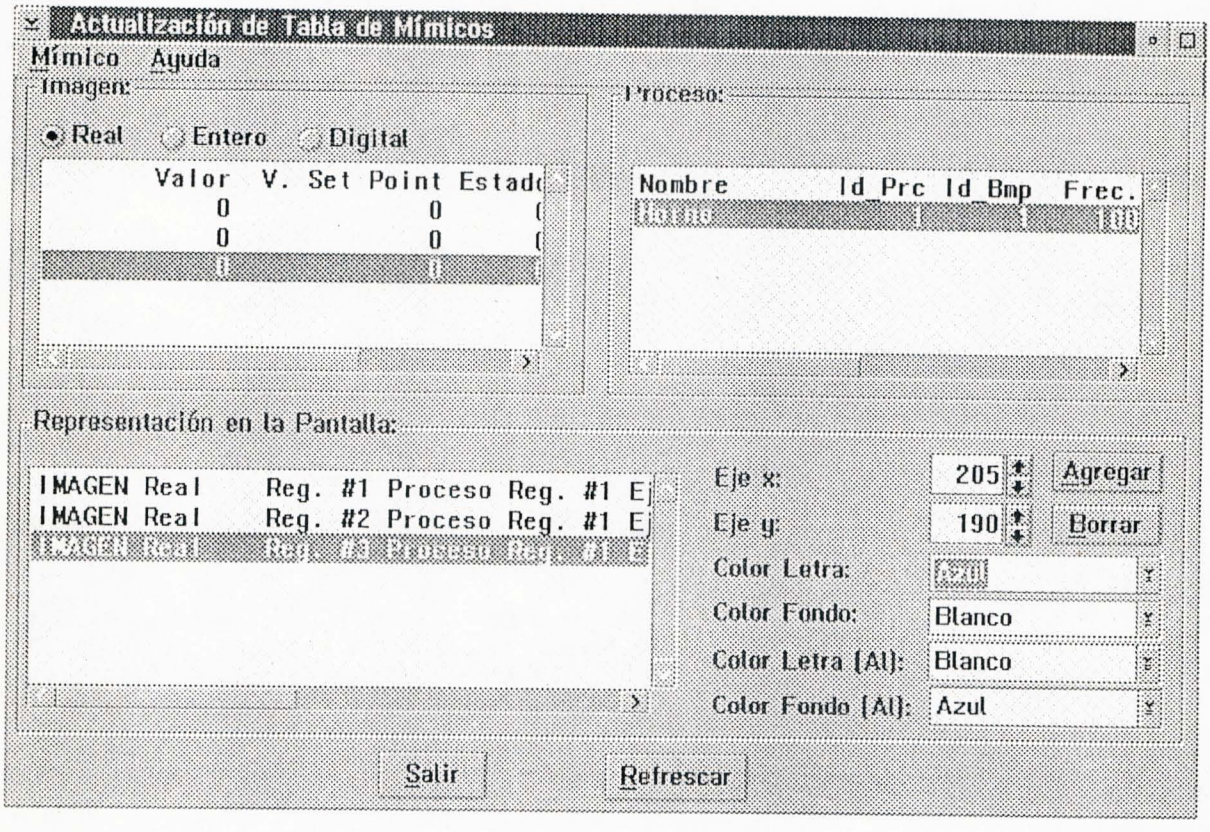
2. **Datos del Proceso:** esta sección está ubicada en la parte derecha central de la ventana. Representa todos los datos de un proceso: Nombre, Identificación del Proceso, Frecuencia, Descripción, Identificación del Bitmap Asociado, Bitmap Asociado. También se encuentran tres objetos de tipo Button para las acciones de *Agregar*, *Modificar* y *Borrar* un proceso.

3. **Proceso en Monitor:** esta sección está ubicada en la parte izquierda superior de la ventana. Es un objeto de tipo EntryField donde se muestran los datos del proceso que se está monitoreando en la Ventana Principal.

4. **Tabla de Procesos:** en la parte izquierda inferior de la ventana se encuentra un objeto de tipo ListBox que contiene todos los procesos definidos por el operador del sistema. Los datos de los procesos se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. Se pueden marcar procesos viendo reflejados sus valores en la sección de Datos del Proceso. El objeto de tipo Button *Refrescar* permite refrescar la información de los procesos definidos. El objeto de tipo Button *Monitorear* marca el proceso seleccionado en la tabla como el proceso en monitor.

3.4 Actualización de la Tabla de Mímicos

En esta ventana se muestran los mímicos definidos por el operador, sus valores y se permite la actualización de los mismos. Es invocada desde la Ventana Principal.



Esta ventana se encuentra dividida en 4 secciones:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: agregar un mímico, borrarlo, refrescar la información, salir, ayuda.

2. **Imagen:** esta sección está ubicada en la parte superior izquierda de la ventana. Allí se encuentran 3 objetos de tipo RadioButton que permiten seleccionar el segmento de la imagen sobre el que se va a operar y 1 objeto de tipo ListBox que contiene todos los puntos de la imagen del segmento seleccionado, que fueron previamente definidos por el operador del sistema. Los datos de los puntos de la imagen se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. En este objeto se debe marcar el punto de la imagen que se quiere representar.

3. **Tabla de Procesos:** esta sección está ubicada en la parte derecha superior de la ventana. Allí se encuentra ubicado un objeto de tipo ListBox que contiene los datos de todos los procesos ingresados por el operador.

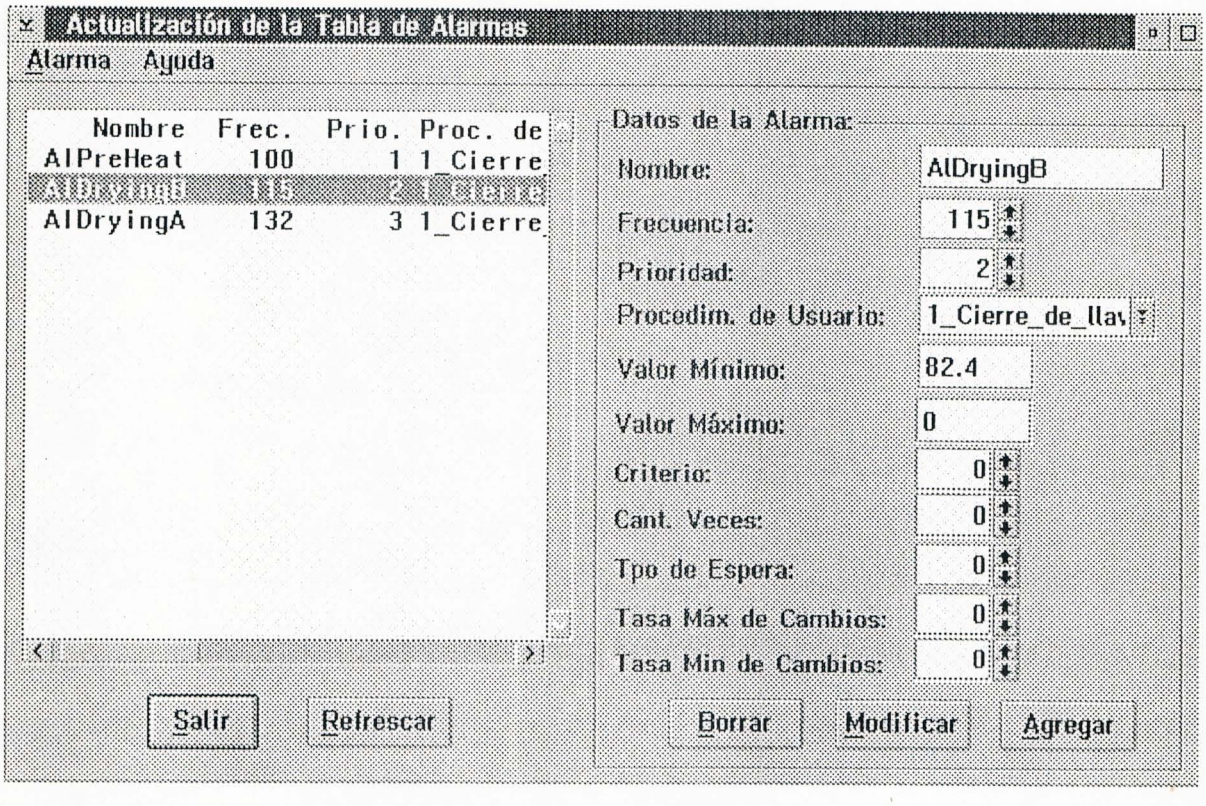
Esta información está organizada de manera tabular y se puede recorrer haciendo scrolling del objeto. En este objeto se debe marcar el proceso con en el cual se quiere representar la información de algún punto de la imagen.

4. **Representación en la Pantalla:** esta sección está ubicada en la parte central de la ventana. Dentro de esta sección, en la parte izquierda se muestra un objeto de tipo ListBox contiene los datos de los mímicos definidos, esta información está presentada de manera tabular y se puede recorrer haciendo scrolling del objeto; en la parte derecha se muestran los atributos de representación de los puntos de la imagen: Eje X, Eje Y, Color de Letra, Color de Fondo, Color de Letra en Estado de Alarma y Color de Fondo en Estado de Alarma. Cada vez que se selecciona un elemento del objeto ListBox, los valores del mímico seleccionado son representados en todas las secciones de la ventana.

En la parte inferior de la ventana, y fuera de las secciones mencionadas anteriormente, se encuentra el objeto de tipo Button *Refrescar*, que refresca la información de la Imagen, Tabla de Procesos y Representación en la Pantalla.

3.5 Actualización de la Tabla de Alarmas

En esta ventana se muestran las condiciones de alarmas definidas por el operador, sus valores y se permite la actualización de las mismas. Es invocada desde la Ventana Principal.



Esta ventana está compuesta por 3 secciones principales:

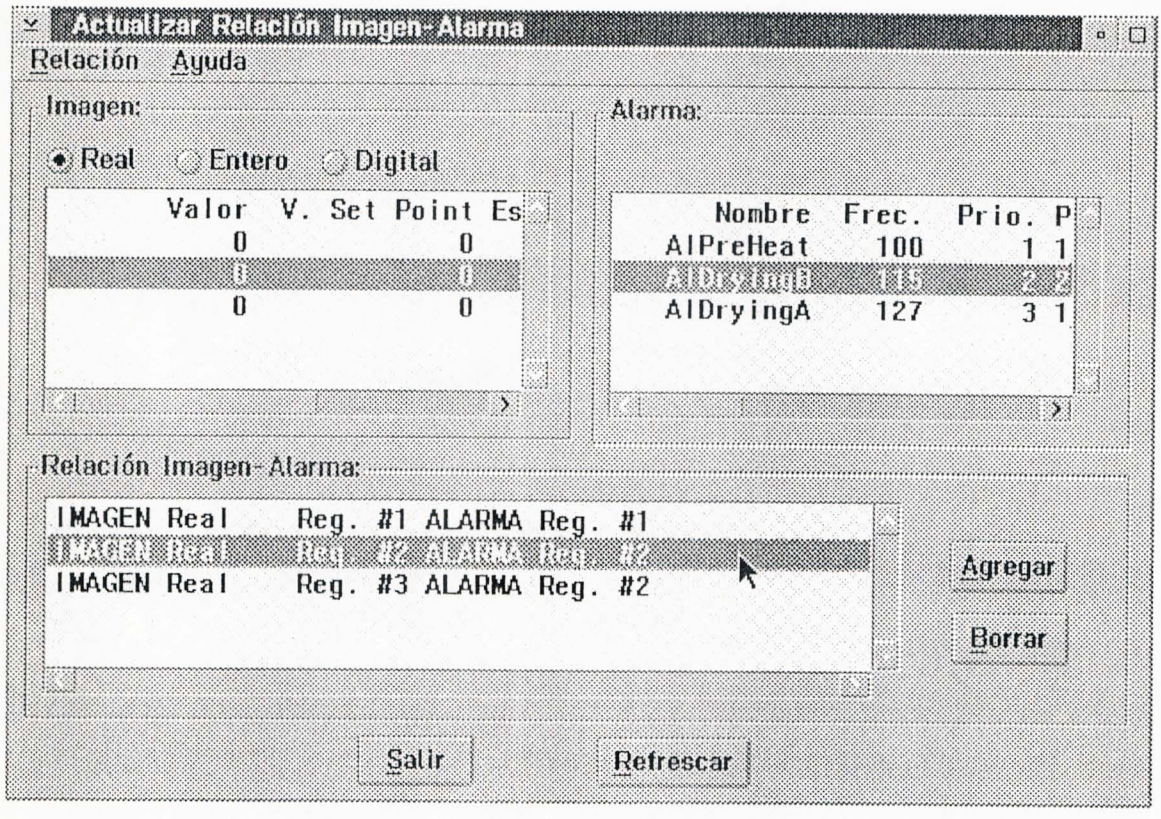
1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: agregar una condición de alarma, modificarla, borrarla, refrescar la información, salir, ayuda.

2. **Datos de la Condición de Alarma:** esta sección está ubicada en la parte derecha central de la ventana. Representa todos los datos de una condición de alarma: nombre, frecuencia, prioridad, procedimiento de atención de alarma, valor mínimo, valor máximo, criterio de evaluación, cantidad de veces que se evaluó, tiempo de espera, tasa mínima de cambios, y tasa máxima de cambios. También se encuentran tres objetos de tipo Button para las acciones de *Agregar*, *Modificar* y *Borrar* una condición de alarma.

3. **Tabla de Alarmas:** en la parte izquierda y central de la ventana se encuentra un objeto de tipo ListBox que contiene todas las condiciones de alarma definidas por el operador del sistema. Los datos de las condiciones de alarma se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. Se pueden marcar condiciones de alarma viendo reflejados sus valores en la sección de Datos de la Condición de Alarma. El objeto de tipo Button *Refrescar* permite refrescar la información de las condiciones de alarma definidas.

3.6 Actualización de la Relación Imagen-Alarma

En esta ventana se muestran las relaciones definidas por el operador entre los puntos de la imagen y las condiciones de alarma y se permite la actualización de las mismas. Es invocada desde la Ventana Principal.



Esta ventana se encuentra dividida en 4 secciones:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: agregar una relación, borrarla, refrescar la información, salir, ayuda.

2. **Imagen:** esta sección está ubicada en la parte superior izquierda de la ventana. Allí se encuentran 3 objetos de tipo RadioButton que permiten seleccionar el segmento de la imagen sobre el que se va a operar y 1 objeto de tipo ListBox que contiene todos los puntos de la imagen del segmento seleccionado, que fueron previamente definidos por el operador del sistema. Los datos de los puntos de la imagen se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. En este objeto se debe marcar el punto de la imagen que se quiere relacionar.

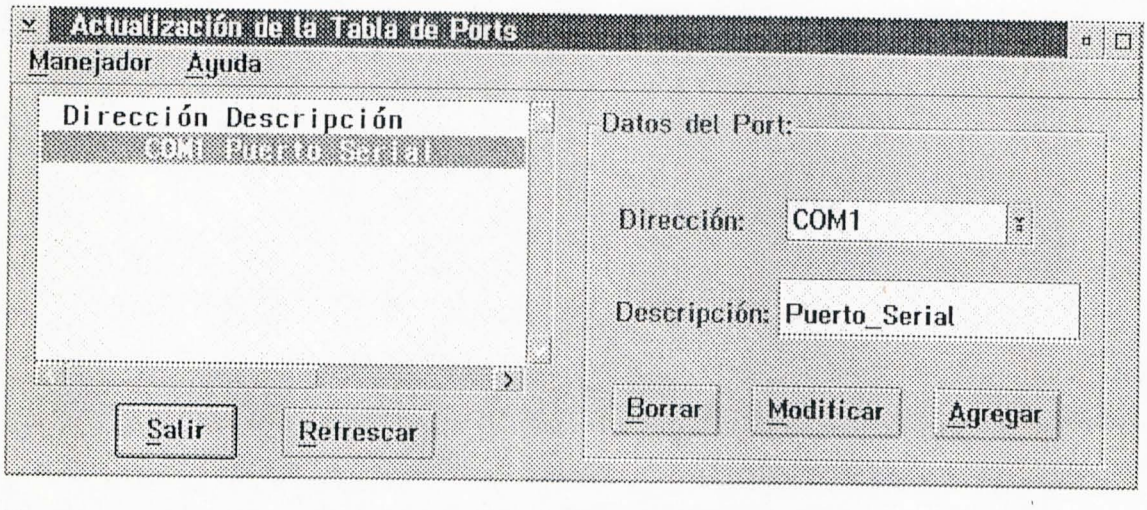
3. **Tabla de Alarmas:** en la parte derecha superior de la ventana se encuentra un objeto de tipo ListBox que contiene todas las condiciones de alarma definidas por el operador del sistema. Los datos de las condiciones de alarma se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. En este objeto se debe marcar la condición de alarma que se quiere relacionar.

4. **Tabla de Relaciones:** esta sección está ubicada en la parte central de la ventana. Dentro de esta sección, en la parte izquierda se muestra un objeto de tipo ListBox que contiene los datos de las relaciones definidas, esta información está presentada de manera tabular y se puede recorrer haciendo scrolling del objeto; en la parte derecha se encuentran los objetos de tipo Button *Agregar* y *Borrar* que sirven para agregar y borrar relaciones. Cada vez que se marca un elemento del ListBox, se marcan automáticamente los elementos de las secciones Imagen y Tabla de Alarmas que componen la relación.

En la parte inferior de la ventana, y fuera de las secciones mencionadas anteriormente, se encuentra el objeto de tipo Button *Refrescar*, que refresca la información de la Imagen, Tabla de Alarmas y Relaciones.

3.7 Actualización de la Tabla de Ports

En esta ventana se muestran los dispositivos externos definidos por el operador, sus valores y se permite la actualización de los mismos. Es invocada desde la Ventana Principal.



Esta ventana está compuesta por 3 secciones principales:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: agregar una definición de dispositivo, modificarla, borrarla, refrescar la información, salir, ayuda.

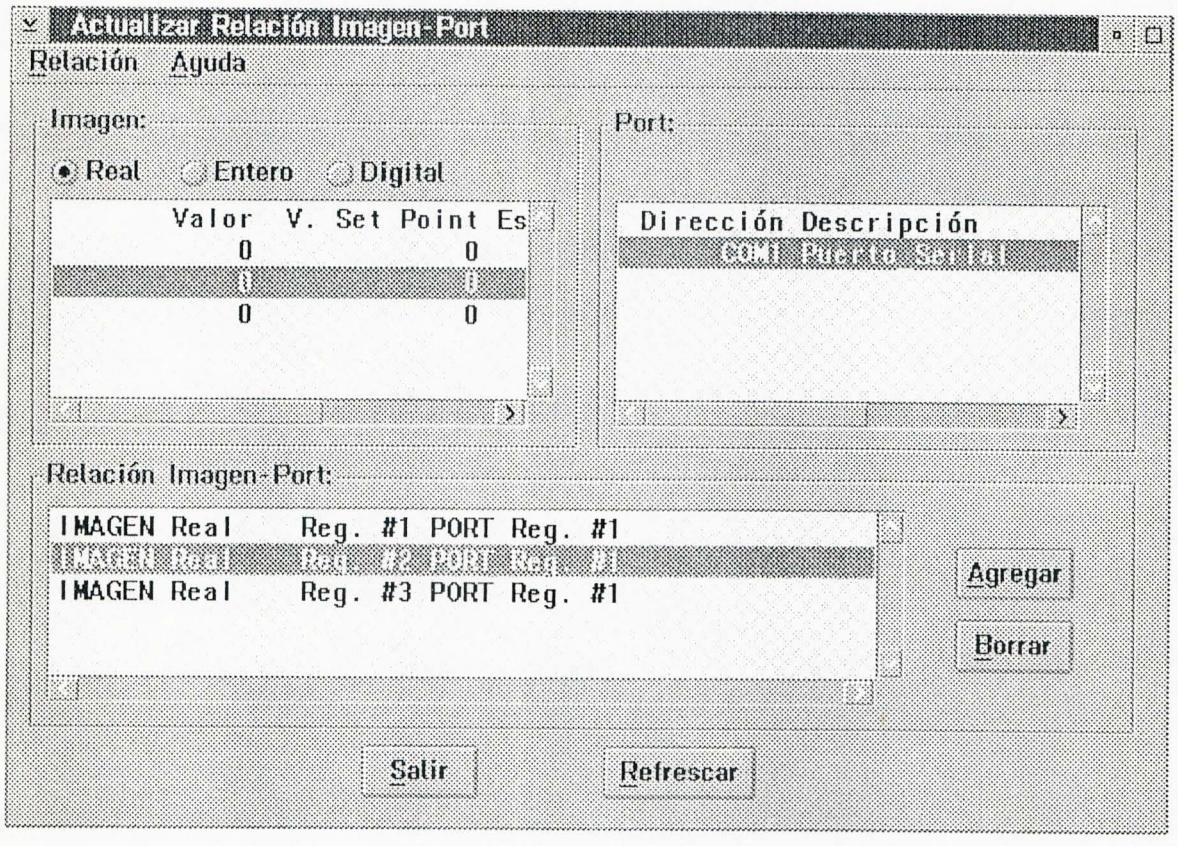
2. **Datos del Dispositivo:** esta sección está ubicada en la parte derecha central de la ventana. Representa todos los datos de una definición de dispositivo: Dirección, Sub-Dirección y Descripción.

También se encuentran tres objetos de tipo Button para las acciones de *Agregar*, *Modificar* y *Borrar* una condición de alarma.

3. **Tabla de Dispositivos:** en la parte izquierda y central de la ventana se encuentra un objeto de tipo ListBox que contiene todas las definiciones de dispositivo definidas por el operador del sistema. Los datos de las definiciones se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. Se pueden marcar definiciones de dispositivo viendo reflejados sus valores en la sección de Datos del Dispositivo. El objeto de tipo Button *Refrescar* permite refrescar la información de las definiciones de dispositivo definidas.

3.8 Actualización de la Relación Imagen-Port

En esta ventana se muestran las relaciones definidas por el operador entre los puntos de la imagen y las definiciones de dispositivos y se permite la actualización de las mismas. Es invocada desde la Ventana Principal.



Esta ventana se encuentra dividida en 4 secciones:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: agregar una relación, borrarla, refrescar la información, salir, ayuda.

2. **Imagen:** esta sección está ubicada en la parte superior izquierda de la ventana. Allí se encuentran 3 objetos de tipo RadioButton que permiten seleccionar el segmento de la imagen sobre el que se va a operar y 1 objeto de tipo ListBox que contiene todos los puntos de la imagen del segmento seleccionado, que fueron previamente definidos por el operador del sistema. Los datos de los puntos de la imagen se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. En este objeto se debe marcar el punto de la imagen que se quiere relacionar.

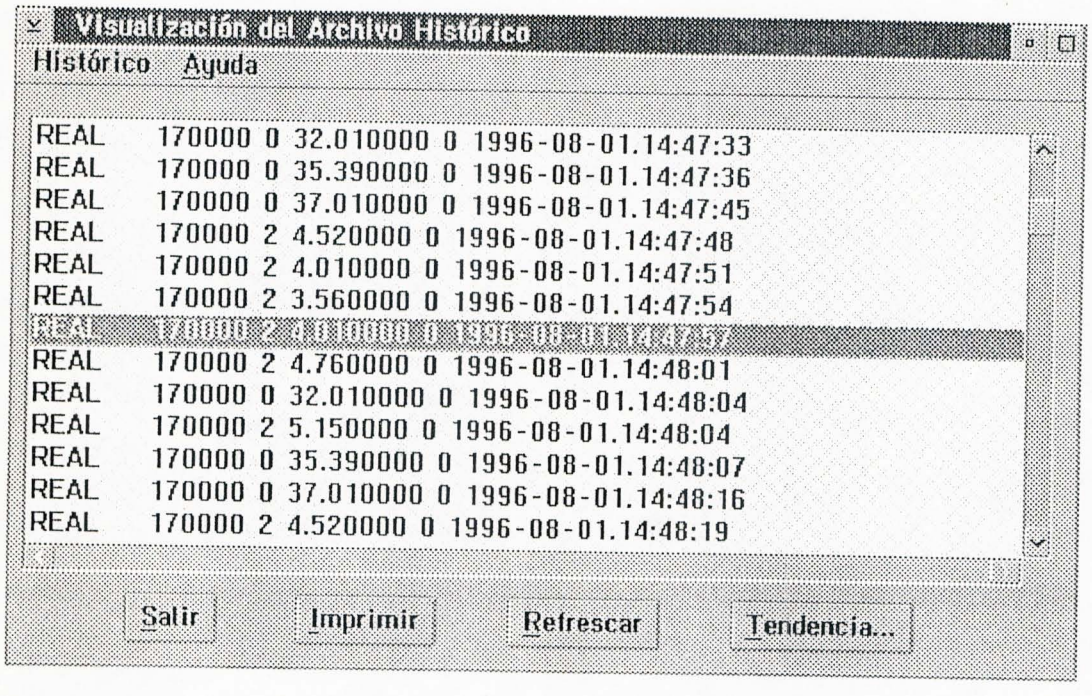
3. **Tabla de Dispositivos:** en la parte derecha superior de la ventana se encuentra un objeto de tipo ListBox que contiene todas las definiciones de dispositivos definidas por el operador del sistema. Los datos de las definiciones se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto. En este objeto se debe marcar la definición de dispositivo que se quiere relacionar.

4. **Tabla de Relaciones:** esta sección está ubicada en la parte central de la ventana. Dentro de esta sección, en la parte izquierda se muestra un objeto de tipo ListBox que contiene los datos de las relaciones definidas, esta información está presentada de manera tabular y se puede recorrer haciendo scrolling del objeto; en la parte derecha se encuentran los objetos de tipo Button *Agregar* y *Borrar* que sirven para agregar y borrar relaciones. Cada vez que se marca un elemento del ListBox, se marcan automáticamente los elementos de las secciones Imagen y Tabla de Dispositivos que componen la relación.

En la parte inferior de la ventana, y fuera de las secciones mencionadas anteriormente, se encuentra el objeto de tipo Button *Refrescar*, que refresca la información de la Imagen, Tabla de Dispositivos y Relaciones.

3.9 Visualización del Archivo Histórico

En esta ventana se muestran los registros históricos almacenados por la aplicación. Es invocada desde la Ventana Principal.



Esta ventana está compuesta por 3 secciones principales:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: imprimir el archivo histórico, refrescar la información, calcular la tendencia de un punto, salir, ayuda.

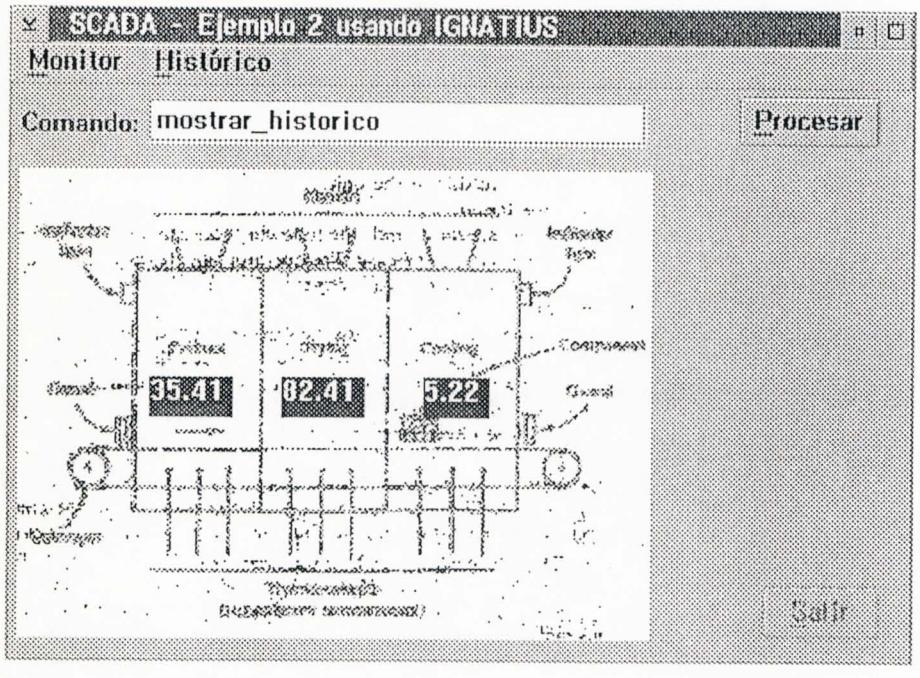
2. **Tabla de Registros Históricos:** en la parte central de la ventana se encuentra un objeto de tipo ListBox que contiene todos los registros históricos grabados por la aplicación. Los datos de las registros se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto.

3. **Botones de Acciones:** en la parte inferior de la ventana se encuentran 4 objetos de tipo Button que permiten realizar las acciones ya mencionadas en la sección Barra de Acciones: *Salir, Imprimir, Tendencia,* y *Refrescar.*

4. Apéndice B - Ejemplo 2 de SCADA que usa IGNATIUS

4.1 Ventana Principal

Esta es la primer ventana que aparece cuando se ejecuta el supervisor



1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana. Estas acciones permiten acceder a las otras ventanas de la aplicación, arrancar el motor de ejecución, detener el motor de ejecución, visualizar el archivo histórico, salir de la aplicación, etc..

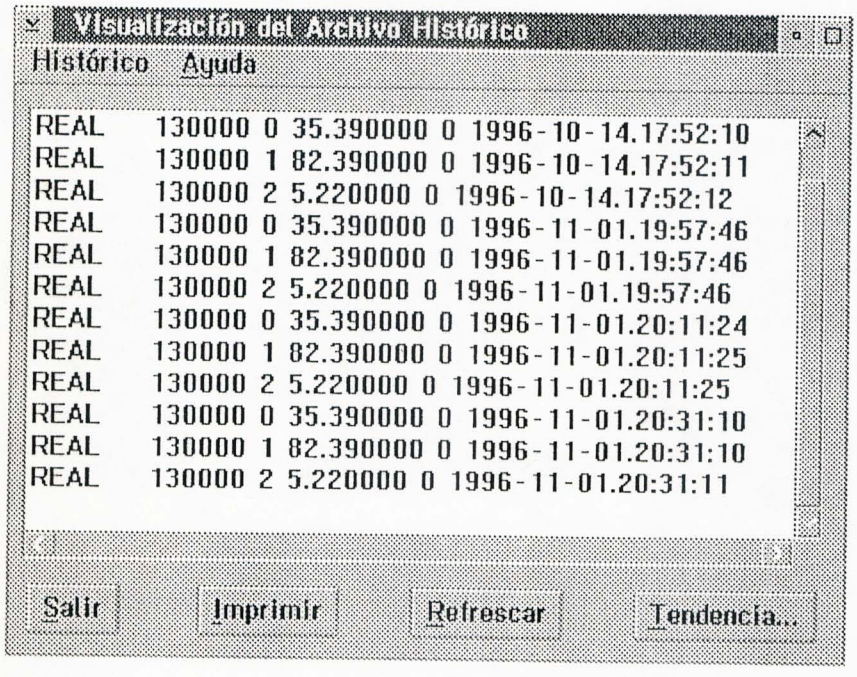
2. **Bitmap de Proceso:** en esta sección se despliega el bitmap del proceso que se está monitoreando. En el se muestran el estados de las distintas alarmas del proceso.

3. **Prompt de Comando:** aquí se ingresan los comandos válidos para la aplicación.

4. **Botón Procesar:** Con este botón se comienza a ejecutar el supervisor.

4.2 Visualización del Archivo Histórico

En esta ventana se muestran los registros históricos almacenados por la aplicación. Es invocada desde la Ventana Principal.



Esta ventana está compuesta por 3 secciones principales:

1. **Barra de Acciones:** en ella se encuentran todas las acciones que se pueden ejecutar en esta ventana: imprimir el archivo histórico, refrescar la información, calcular la tendencia de un punto, salir, ayuda.

2. **Tabla de Registros Históricos:** en la parte central de la ventana se encuentra un objeto de tipo ListBox que contiene todos los registros históricos grabados por la aplicación. Los datos de las registros se visualizan de manera tabular y se puede recorrer esta información haciendo scrolling del objeto.

3. **Botones de Acciones:** en la parte inferior de la ventana se encuentran 4 objetos de tipo Button que permiten realizar las acciones ya mencionadas en la sección Barra de Acciones: *Salir, Imprimir, Tendencia,* y *Refrescar.*