Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Software para el cálculo y la simulación del modelo de un evaporador en simple efecto

Autor: Arian E. Calzadilla Cabrera

Tutor: Ing. María del Carmen Hernández Carús

Santa Clara

2004

"Año del 45 Aniversario del Triunfo de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Software para el cálculo y la simulación del modelo de un evaporador en simple efecto

Autor: Arian E. Calzadilla Cabrera

E-mail: calzadilla@uclv.edu.cu

Tutor: Ing. María del Carmen Hernández Carús

Prof. Auxiliar, Depto. Automática y Sistemas Computacionales

Facultad Ing. Eléctrica

E-mail: carmen@uclv.edu.cu

Santa Clara

2004

"Año del 45 Aniversario del Triunfo de la Revolución"

Hago constar que el presente trabajo fue realizado en la Universidad Central "Marta Abreu" de las Villas (UCLV) como parte de la culminación de los estudios de la carrera de Ingeniería en Automática. Autorizo a que el mismo sea utilizado por dicha institución para los fines que estime conveniente tanto de forma parcial como total, y que además no pueda ser presentado en eventos, ni publicado, sin la autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado con la aprobación de la Dirección de nuestro centro y que el mismo cumple con los requisitos que debe tener un trabajo de esta naturaleza, referido a la temática señalada.

_____ _____

Firma del Tutor Firma del Jefe de Dpto.

_____

Firma del Responsable de
Información Científico-Técnica

Pensamiento

“Todos somos aficionados. En esta corta vida no da tiempo a otra cosa”

Charles Chaplin

Agradecimientos

A mis padres, por su preocupación durante toda la vida.

A mi amigo Rodney Hernández, por su invaluable ayuda.

A mi tutora María del C. Hernández, por su ayuda incondicional, por la confianza.

A mis amigos; por hacerme, quizás sin yo decirlo, la vida más agradable.

A los que no considero amigos, pero que de una forma u otra me ayudaron.

Dedicatoria

A mis padres

A mi Abuela querida

A mi único hermano, como un intento más…

Tarea Técnica

TAREA TECNICA

Tareas parciales a realizar para culminar el trabajo.

1. Establecer los requisitos de la aplicación a desarrollar.
2. Consultar la bibliografía especializada y trabajos realizados en la facultad para desglosar las especificaciones del modelo en el espacio de estado que describe a un evaporador.
3. Investigar acerca de las posibilidades de comunicación de MATLAB con otras aplicaciones.
4. Confeccionar un prototipo de la aplicación para la comprobación de su eficacia en una clase de laboratorio de la asignatura Procesos. Corregir errores encontrados.
5. Análisis de los resultados.
6. Redacción del informe final.

Resumen

RESUMEN

En la especialidad de Automática y Sistemas Computacionales es ampliamente utilizada la herramienta de software MATLAB, que constituye un poderoso instrumento para el cálculo y la simulación de modelos matemáticos complejos. Desde hace ya varios años se ha convertido en el programa de simulación seleccionado por todos los autores de bibliografía especializada en el tema de Control, para mostrar ejemplos y sugerir estudios independientes.

En varias asignaturas de la carrera, donde se analizan y diseñan lazos de control, la planta en cuestión puede estar representada por modelos matemáticos de mayor o menor complejidad. En el caso específico de los llamados Procesos, la obtención de dichos modelos puede tornarse engorrosa en función de su complejidad y de la cantidad de variables y parámetros que se involucren. Tal es el caso del proceso de evaporación. Si bien es cierto que la estructura del modelo ha quedado determinada hace años, los valores de los parámetros de este, deben obtenerse haciendo numerosos cálculos y operaciones matemáticas. Esta operación en MATLAB si bien no es imposible, es tediosa porque la interfaz básica de usuario que ofrece, es un intérprete de comandos de texto. Si se dispone de una herramienta computacional que automatice el proceso para obtener los parámetros del modelo, entonces el estudiante agilizará su trabajo y podrá dedicar mayor tiempo al análisis del comportamiento de la planta, seleccionar y diseñar los lazos, etc. En función de tratar de garantizar esto último es que se enmarcan nuestros esfuerzos en este trabajo.

Índice

INDICE

Introducción

Introducción

En asignaturas terminales de la carrera de Automática y Sistemas Computacionales, el análisis y el diseño de sistemas multivariables son objetivos esenciales a cumplir. De forma general, para el análisis y/o diseño de cualquier sistema se parte de un modelo conocido que lo describa.

La modelación de plantas con distintos grados de complejidad, se aprende en varias asignaturas de la disciplina de Sistemas de Control. Comienza en el tercer año con la asignatura Modelación y Simulación; sin embargo es en la asignatura Procesos cursada en el cuarto año donde se profundiza en la obtención de modelos de plantas complejas como son: generadores de vapor, intercambiadores de calor, columnas de destilación, evaporadores, entre otros. La obtención de un modelo que refleje la realidad de manera más o menos apropiada y confiable para el análisis y diseño de un sistema con estas características, es siempre una tarea difícil y tediosa, dado el gran número de ecuaciones necesarias para describir su funcionamiento.

El caso que nos ocupa en este trabajo es un ejemplo de ello. El modelo en el espacio de estado de un evaporador de calandria vertical en simple efecto, tiene 4 variables de salida y 5 de entrada. Por la complejidad del proceso en sí mismo es necesario, para modelarlo, tener en cuenta un gran número de parámetros y condiciones de operación. Por ser una planta usada comúnmente en la industria y constituir un ejemplo de sistema multivariable complejo, con un modelo ampliamente descrito en la literatura y abundado en trabajos de diploma anteriores de graduados de nuestra especialidad, se lo escoge como caso típico de estudio en nuestra carrera.

Debido a las enormes potencialidades que ofrecen los sistemas computacionales de cálculo y simulación disponibles hoy en día, son incluidas en la enseñanza de las asignaturas de la disciplina de Sistemas de Control, actividades de laboratorio donde el alumno emplea los conocimientos adquiridos para diseñar y comprobar a través de la simulación en softwares apropiados, la solidez de sus conocimientos. MATLAB es una herramienta computacional de amplio uso a nivel mundial en sectores tan diversos como el aeroespacial, la defensa, la automoción y las comunicaciones, rebasando con creces los requerimientos de aplicabilidad que establecen casi todos los aspectos del Control Automático. Sus potencialidades lo

hacen objeto de referencia para explicar de manera práctica muchos ejemplos de la literatura especializada empleada en nuestra enseñanza y constituye la solución escogida para numerosos proyectos desarrollados por la comunidad científica actual en materia de Automática. La selección de MATLAB como herramienta de simulación en las facultades que imparten nuestra carrera en Cuba, por tanto, no responde a un hecho fortuito, sino al consenso casi generalizado en la opinión de que es lo mejor que ofrece el mercado para apoyar los trabajos en los temas que ocupan a nuestra especialidad.

Para iniciar la simulación en MATLAB del modelo en el espacio de estado referenciado en el presente trabajo [1] basta con proveer a este las matrices que lo describen. Pero el proceso de transformación que ocurre a partir de la entrada de los datos iniciales hasta la obtención de estas matrices es la parte que, utilizando las interfaces básicas de usuario que ofrece MATLAB, puede tornarse engorrosa. Pues el modo estándar de comunicación de este programa con el usuario es un intérprete de comandos de texto. También existe la posibilidad de generar un *script* que contenga toda la lista de comandos requeridos para realizar dichas transformaciones, la cual es ejecutada en forma lote cuando se llama al *script* desde la ventana principal. Pero este es un método que por inflexible y poco intuitivo, hace perder el interés al alumno y consumir mucho tiempo en aspectos ajenos a la asimilación del conocimiento derivado de la simulación. Es aquí donde aparece el problema a resolver en nuestro trabajo.

Se pretende la confección de una aplicación, que automatice el proceso de obtención de las matrices que describen el modelo en el espacio de estado de un evaporador de calandria vertical. Haciendo transparente al usuario los cálculos intermedios que no requieran de su participación. Solo serían necesarios los datos indispensables para calcular todos los parámetros del modelo, los cuales serían provistos por el usuario a través de una interfaz gráfica cómoda y didáctica. Luego, las matrices de estado se exportarían a MATLAB para comenzar la simulación propiamente dicha.

Con la interposición de este elemento en la comunicación Usuario-MATLAB, disminuye considerablemente la cantidad de tiempo que media entre la identificación de los datos concretos de funcionamiento del proceso en un momento dado, y la simulación de su comportamiento. Dándole la oportunidad al alumno de dedicar mas esfuerzos a investigar la aplicación de las distintas técnicas de control para diseñar los lazos que se requieran. Una

utilidad adicional de este proyecto es que deja a la asignatura Procesos mejor preparada para asumir la adopción del nuevo Plan de Estudios D donde se pretende el aumento de las horas dedicadas al trabajo independiente del alumno. Haciendo más autodidacta la asimilación de los conocimientos relacionados con este tema, se requiere mucho menos de la supervisión del aprendizaje del alumno por parte de su profesor.

Capítulo 1

1. CONSIDERACIONES SOBRE LA OBTENCION DE LOS PARAMETROS DEL MODELO DEL EVAPORADOR, CONOCIDA SU ESTRUCTURA, Y LA SIMULACION DEL MISMO UTILIZANDO MATLAB

1.1 Introducción

Los evaporadores son equipos muy utilizados en la industria química pues su propósito es tan universal que se le puede encontrar formando parte de cualquier flujo productivo. Su presencia en el ámbito nacional es muy amplia, fundamentalmente en la industria azucarera. El principal objetivo del evaporador es el aumento de la concentración de la solución que circule como corriente de alimentación, aunque, además se utilizan con objetivos colaterales como el de la producción de vapor. Como sistema ha sido objeto de investigación desde hace muchos años, quedando detalladamente establecidos en la literatura, entre otros aspectos, los procesos físicos que describen su comportamiento.

En la asignatura Procesos de la carrera de Automática, junto a otros procesos multivariables que se modelan, se estudia el de los evaporadores que más comúnmente pueden encontrarse en la industria azucarera de nuestro país. Se toma como punto de partida la modelación realizada desde hace varios años por Aguado y sus colaboradores [1] porque se considera que dicho modelo representa de forma verídica el comportamiento dinámico de las variables que se involucran en el proceso de evaporación. Usualmente se orienta el estudio individual del procedimiento matemático seguido en este trabajo, y a partir de un conjunto de datos de un caso práctico, el estudiante debe llegar al modelo final. Con el objetivo de mostrar la complejidad de esa tarea se expondrán brevemente a continuación las características de este proceso.

1.2 El modelo del evaporador en simple efecto.

Partiendo de una representación esquemática del evaporador y la definición de sus variables y parámetros, como se muestra en la Fig.1:

Donde:

P_1 - Presión en la cámara superior.

P_2 - Presión en la cámara inferior.

C_2 - Concentración del líquido de entrada.

C_1 - Concentración del líquido evaporado.
M_1 - Flujo de líquido de entrada.
M_2 - Flujo de líquido evaporado.
M_s - Flujo de entrada de vapor de calefacción.
S_0 - Posición de la válvula que controla la salida del líquido evaporado.
h - Nivel de líquido dentro del evaporador.

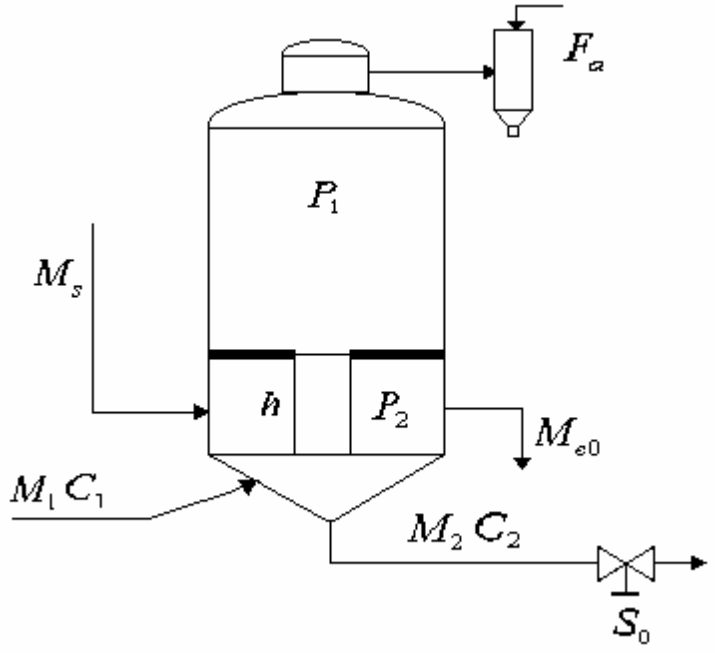

Fig.1 Esquema de un evaporador.

Sin profundizar de manera alguna en la modelación (porque no constituye objetivo de este trabajo), la cual puede encontrarse en la literatura [1] y en trabajos de diploma de años anteriores [6], explicaremos brevemente la metodología que se sigue para obtener el modelo.

Primeramente se construyen una gran cantidad de ecuaciones para describir el comportamiento dinámico de un evaporador en simple efecto de calandria vertical, según las consideraciones expuestas en [6] para lograr un modelo que describa el sistema en el espacio de estado, el autor determina la necesidad de reducir el número de variables que integran dichas ecuaciones y también el número de estas. Esto se logra a través de los procesos de normalización y linealización, asegurando que si se permiten solo pequeñas desviaciones de las variables con respecto al valor nominal de operación las ecuaciones modificadas continúan describiendo bastante fielmente la dinámica del proceso. Luego,

haciendo sustituciones progresivas, se llega a plantear un modelo que describe bastante fielmente la dinámica del proceso con un número mínimo de ecuaciones que nos lleve a la representación en el espacio de estado.

Es válido destacar que la forma de escoger las variables del sistema no es única; por lo que en la selección idónea de estas, interviene la experiencia del diseñador del sistema, además de las observaciones de índole tecnológica necesarias. Finalmente, el modelo en el espacio de estado, utilizado en nuestro trabajo, que describe el comportamiento de un evaporador de calandria vertical es el que sigue:

$$\dot{x} = \mathrm{A}x + \mathrm{B}u + \mathrm{F}v$$
$$y = \mathrm{C}x$$

Donde:

$x = (\mathrm{h},\ P_1,\ P_2,\ C_2)$ - Vector de estado
$u = (M_s,\ S_0, F_a)$ - Vector de control
$v = (M_1, C_1)$ - Vector de perturbaciones
$y = (\mathrm{h},\ P_1,\ P_2,\ C_2)$ - Vector de salida
A, B, F y *C* - Matrices de estado.

La inclusión de este complejo sistema como caso de estudio para la enseñanza de las asignaturas que integran la disciplina de Sistemas de Control en las carreras de Automática del país, responde al enorme potencial pedagógico que le confieren la gran cantidad de literatura dedicada al respecto, la variedad de estrategias de control aplicables una vez que se alcanza el modelo, y su importancia estratégica en la industria nacional. Esto hace que continuamente se verifiquen los procedimientos didácticos disponibles por los profesores para garantizar el máximo aprovechamiento de las horas clase dedicadas al estudio teórico y, fundamentalmente práctico, del mismo a través de la simulación.

1.3 La simulación asistida por computadora

El alcance logrado por las técnicas computacionales en nuestros días hace inconcebible la confección de un programa docente dedicado a la enseñanza de sistemas complejos, sin horas dedicadas a la simulación asistida por computadoras. El valor de la simulación es incuestionable ya que nos permite obtener resultados relativos al comportamiento en la

realidad, del objeto simulado, sin empeñar en ello ningún recurso de fabricación ni de implementación práctica. El desarrollo de los medios actuales (*software* y *hardware*) para la simulación de sistemas de control hacen de esta tarea un proceso con un consumo de tiempo razonablemente corto, lo cual la hace un instrumento poderoso en extremo a la hora de abordar problemas simples y complejos que envuelvan al Control Automático. Los estudiantes de nuestra especialidad hacen uso de este método en el programa de estudios, beneficiándose con las lógicas ventajas que ofrece el hecho de poder corroborar de inmediato la eficiencia de las estrategias empleadas para el control de los sistemas tratados en clases. La aplicación de los conocimientos acumulados en las materias de la especialidad, en cualquier estadío de esta, para el diseño de cualquier sistema con la seguridad de su utilidad práctica, deja al alumno en mejores condiciones para transitar por los sucesivos ciclos de asignaturas, que lo forman como profesional especializado apto para enfrentar la práctica con mayores probabilidades de dominarla.

Habíamos hablado anteriormente del binomio necesario para desarrollar cualquier actividad relacionada con el diseño y la simulación asistida por computadoras: *software-hardware*. El componente de *hardware* no es en estos tiempos el mayor problema. Una computadora personal con un procesador capaz de realizar 2000 MIPS (Millones de Instrucciones Por Segundo) con operandos enmarcables en hasta 32 bits de información, monitor capaz de 1024 x 760 *pixels* de resolución gráfica, memorias con tiempos de acceso reducidos y bajo consumo de energía, no rebasa los 1000 dólares en el mercado. Lo cual significa que para la mayoría de las tareas de simulación relacionadas con el Control Automático la diferencia en el rendimiento la impone el componente de *software* empleado a tal efecto. Existen varias opciones en el mercado que pueden suplir las diversas necesidades planteadas por prácticamente cualquier reto que pueda surgir en el quehacer ingenieril de hoy. Dígase por ejemplo: Matematica®, Mathcad®, MATLAB®. Solo que para el caso especifico del Control Automático, esta última opción es la que ha ganado la aceptación de la casi totalidad de las autoridades en esta materia. Ganándose el prestigio suficiente como para prevalecer como herramienta de modelación y simulación preferida por los sectores académicos y productivos, no solo de nuestra especialidad, sino de disímiles áreas del conocimiento.

1.3.1 MATLAB como herramienta para la simulación de sistemas de control.

MATLAB [17] es el nombre abreviado de "MATrix LABoratory". Este es un programa para cálculo técnico y científico que integra cómputo, visualización y posibilidad de programar en un lenguaje propio.

Los usos típicos de este software son:

- Cálculos numéricos y labores de cómputo.
- Desarrollo de algoritmos
- Modelación, simulación y desarrollo de prototipos
- Análisis, exploración y visualización de datos
- Graficado ingenieril y científico

MATLAB es un sistema cuya estructura de dato básica es un arreglo dinámico, o sea, que no requiere dimensionamiento. Acoge a toda una familia de subprogramas de aplicación específica llamados *toolboxes,* los cuales permiten al usuario aprender y aplicar tecnologías especializadas. Los *toolboxes* son grandes conjuntos de funciones desarrolladas en el lenguaje propio de MATLAB. Algunas de las áreas que abarcan los *toolboxes* disponibles son: Procesamiento de Señales, Sistemas de Control, Redes Neuronales, Lógica Fuzzy entre otros.

Se puede decir que todo el sistema MATLAB consta de 5 partes fundamentales:

1. El lenguaje MATLAB: Este es un lenguaje de alto nivel capaz de manipular estructuras de datos complejas. Posee además capacidades para la programación orientada a objetos.
2. El entorno de trabajo de MATLAB: Incluye facilidades para manipular las variables declaradas en el *workspace* e importar y exportar datos. Tiene también herramientas para crear, depurar y manipular los ficheros que conforman las aplicaciones MATLAB.
3. Sistema gráfico: Contiene comandos de visualización gráfica de datos en 2-D y 3-D, asi como para el procesamiento de imágenes.
4. La librería de funciones matemáticas de MATLAB.
5. La interfaz externa para aplicaciones de MATLAB. Esta es un conjunto de capacidades que ofrece el software para permitir la interacción de programas independientes con él o viceversa.

MATLAB posee un *toolbox* llamado *Control System Toolbox,* el cual incluye los métodos clásicos y modernos de diseño de sistemas de control, incluidos el Lugar Geométrico de las Raíces, la colocación de polos y el diseño de reguladores LQG. Las tareas típicas de la Ingeniería de Control se simplifican con interfaces gráficas de usuario apropiadas. Posee una amplia colección de comandos y funciones que tienen una gran utilidad en la realización de tareas típicas realizadas en el análisis y diseño de sistemas. Pueden nombrarse algunas como: graficación del lugar geométrico de las raíces, graficación de la respuesta en frecuencia (Bode y Nyquist), transformación entre modelos en el espacio de estado y función de transferencia, conversión de modelos en espacio de tiempo continuo, a modelos en tiempo discreto, entre otras que abarcan todo el ámbito del diseño.

Un subprograma esencial de MATLAB es el Simulink. Este es un sistema interactivo para la simulación de sistemas. El programa consta de una interfaz gráfica, conducida a través del *mouse* que permite modelar cualquier sistema dibujando diagramas de bloques en la pantalla, los cuales pueden ser modificados dinámicamente. Es aplicable a sistemas lineales, no lineales, continuos en el tiempo, discretos y multivariables. El conjunto de componentes incluidos junto al programa, consta de bibliotecas de fuentes de señal, dispositivos de presentación de datos, conectores, funciones matemáticas, sistemas lineales y no lineales, y diferentes formas de llevar a cabo las simulaciones pudiendo escoger entre diferentes modelos.

Estas son las cartas de presentación de MATLAB como software para la modelación y la simulación de sistemas. Razones más que suficientes como para seguir la tendencia mundial y acogerlo en las facultades que imparten la Ing. Automática en Cuba como herramienta especializada para las actividades docentes de laboratorio e investigación. Sin embargo, a pesar de las indiscutibles cualidades de este programa como *engine* de cálculo, las interfaces básicas de intercambio de información en el sentido Usuario–MATLAB son pobres (estamos hablando de la aplicación principal, sin hacer uso de ningún *toolbox* con ambientación gráfica de propósito específico). Reducidas a una ventana de comandos de texto y a la posibilidad de generar *scripts* en el lenguaje propio. Naturalmente, lo que sucede es que estamos hablando de una herramienta prácticamente con fines universales, a la cual es imposible dotar con una interfaz de usuario que se adapte a las múltiples necesidades de cada especialidad. No es menos cierto que se pueden fabricar interfaces

visuales aceptables haciendo uso del lenguaje que incorpora MATLAB, pero no constituyen aplicaciones independientes que solo tengan una mera relación de comunicación con este, sino que dependen enteramente del mismo para existir. Además de que los controles visuales que ofrece la programación con este lenguaje, para la realización de una interfaz gráfica determinada, no son de la variedad y/o versatilidad requeridas para una aplicación que pudiera exigir dichos requisitos.

1.4 Uso de MATLAB para obtener el modelo y simular el comportamiento de un evaporador

Tomemos como ejemplo la realización de la tarea extraclase de la asignatura Procesos, cursada en cuarto año del plan de estudios de Ing. Automática. Donde los alumnos deben obtener y simular el modelo de un evaporador de calandria vertical utilizando MATLAB, para luego analizar y comentar los resultados obtenidos.

Retomando el modelo del evaporador, mencionado en el epígrafe 1.2, se observa que las matrices que lo definen tienen la forma que se expresa a continuación:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} \qquad B = \begin{bmatrix} 0 & b_{12} & 0 \\ 0 & 0 & b_{23} \\ b_{31} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$F = \begin{bmatrix} f_{11} & f_{12} \\ 0 & 0 \\ 0 & 0 \\ f_{41} & f_{42} \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El cálculo de cada uno de los elementos distinto de 0 en esas matrices se realiza a través de expresiones determinadas por el autor del modelo y que son función de los parámetros de operación del proceso, constantes físicas, datos tabulados hallados en la literatura o variables propiamente dichas. Son, en su mayoría, expresiones largas y constan de un promedio de 12 términos cada una. No todos los elementos de dichas expresiones necesitan

ser provistos por el estudiante para calcular el modelo del evaporador. Estos elementos se pueden dividir en dos grupos fundamentales: Un juego inicial de datos y otro conformado por aquellos que son derivación de estos, es decir, son producto de alguna transformación donde los datos iniciales constituyen el punto de partida para su obtención. Convengamos en llamar al primer grupo de datos: **datos iniciales** y al segundo: **datos complementarios**.

Por ejemplo, tomemos de [1], el caso de la dependencia entre la densidad del líquido de entrada a un evaporador y su concentración. A pesar de no ser una dependencia lineal, esta se puede considerar como tal para pequeñas desviaciones de su valor nominal. Siendo den_1 la densidad del líquido de alimentación, y C_1 la concentración (ver Fig.1), se puede escribir:

$$den_1 = d_1 C_1 + e_1$$

Esta relación está contenida en tablas de densidad contra concentración, diferentes para cada clase de líquido o fluido. El término d_1 es la derivada de la función de dependencia en el punto que tiene como abscisa C_1. O sea, que C_1 estaría incluido dentro del grupo de los **datos iniciales** y, den_1 y d_1 serian derivación de este, integrando así el grupo de los **datos complementarios**. Obsérvese que las operaciones de transformación para obtener un dato perteneciente al segundo grupo, a partir de otro incluido en el primero, implican un grado de complejidad no despreciable. Este procedimiento que acabamos de describir se repite para la obtención del resto de los **datos complementarios**, los cuales alcanzan el número de 11.

El cálculo de los coeficientes de las matrices de estado es solo una parte de las etapas que debe seguir el alumno para cumplir las exigencias de la tarea. Ya que este paso constituye solamente un proceso intermedio para lograr simular el comportamiento del sistema estudiado. Una vez halladas las matrices que representan el modelo en el espacio de estado, se procede a la obtención de la representación por matriz dinámica haciendo uso de funciones específicas de MATLAB. Cada elemento de dicha matriz es la función transferencia entre una de las salidas establecidas en el modelo y una de las entradas (ya sean pertenecientes al vector de mando o al vector de perturbaciones) también establecidas

en el modelo. Es preciso hallar dos matrices dinámicas, ya que el vector de entrada de una es el vector de mando del modelo y el otro es el de las perturbaciones. Luego se procede a simular la respuesta ante un estímulo definido, de cualquier par entrada-salida establecido en el modelo. La información sobre el comportamiento del sistema es un gráfico con la respuesta en el tiempo.

Para llevar a cabo todas las operaciones de la metodología descrita anteriormente, el estudiante tiene ante sí dos posibles vías para ingresar los **datos iniciales** en MATLAB y comenzar así la secuencia de pasos requeridos para lograr sus objetivos.

Interactuando con la ventana de comandos o *command window,* se hace cargo de una labor tediosa ya que tiene que ejecutar uno por uno los comandos (con el consiguiente gasto de tiempo) necesarios para entrar los **datos iniciales** y luego hacer las transformaciones necesarias para obtener finalmente la matriz dinámica del sistema. Luego puede simular el par entrada–salida que desee, lo cual significa, a pesar de constituir otra secuencia de comandos, un esfuerzo ciertamente despreciable comparado con el realizado en el proceso anterior.

En cambio, si el usuario decidiese elaborar un archivo *.m* para acelerar el procedimiento de cálculo, obtendría ventajas respecto al método anterior. Ya que a través de esta vía puede declarar variables con los **datos iniciales** y escribir a continuación toda la secuencia de comandos para seguir la metodología descrita, como una lista de órdenes que, al cargar el fichero, se ejecutan de una vez. Debe tenerse el cuidado de ordenar correctamente la secuencia de los comandos declarados en el fichero *.m* para que las variables utilizadas en los lugares donde correspondan dentro del *script* se hallen ya declaradas en el *workspace*. A pesar de significar un avance sobre el método anterior, esta solución presenta sus inconvenientes ya que no es del todo factible, pues cada vez que los datos del problema varíen, será necesario editar el *script* dando lugar a la posible introducción de errores dentro del archivo. Además de hacer requerir el uso de un editor de texto, para este fin.

Los profesores a cargo de esta tarea extraclase observan que a lo largo de su realización, los estudiantes se debaten en una serie de aspectos netamente procedimentales a la hora de realizar los ejercicios que les son indicados. Y no consiguen tener listo el modelo lo suficientemente pronto como para dedicar suficiente tiempo a analizar los aspectos derivados de la fase de simulación del diseño. Ya sea por la falta de práctica de los

estudiantes manipulando el software, o por lo engorroso del procedimiento de cálculo, la tarea no cumple a cabalidad extrema con el objetivo de consolidar los conocimientos adquiridos en las conferencias y clases prácticas. A raíz de este problema es que surge la necesidad de resolver, de algún modo, esta dificultad en el correcto desarrollo de las actividades de trabajo independiente. La Fig.2 muestra las etapas del proceso actual de cálculo y simulación donde el alumno debe escribir comandos en MATLAB para poder vencerlas.

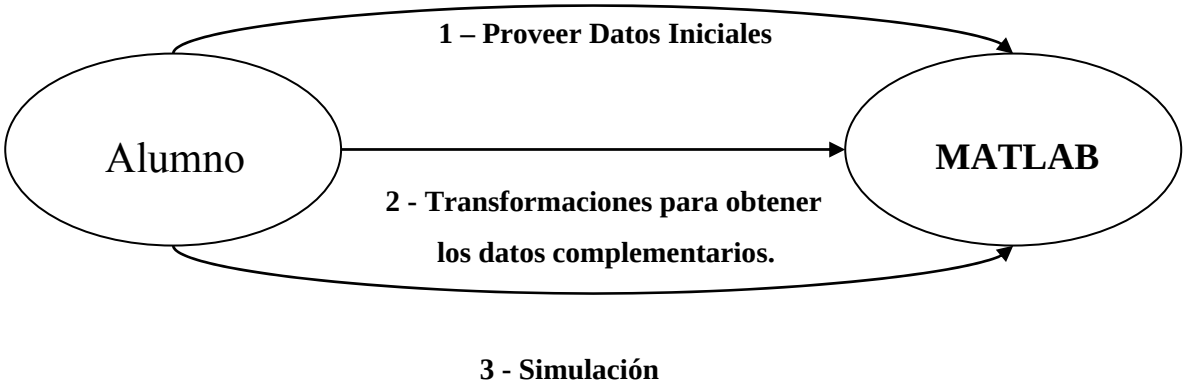

Fig.2: Etapas del proceso de cálculo y simulación donde el alumno debe formular comandos escritos, según los métodos actuales.

1.5 Variante para mejorar la comunicación Alumno-MATLAB en la obtención y simulación del modelo del evaporador

Si se interpusiera un elemento en el canal de comunicación Alumno–MATLAB, de forma tal que lo libere de generar los comandos para realizar las transformaciones necesarias en la obtención de los **datos complementarios** y dar comienzo a la simulación; quedando solamente a cargo de brindarle los **datos iniciales** a este elemento intermedio, la automatización del procedimiento seria más abarcadora y disminuirían así las posibilidades de errores humanos en aquellas tareas no sujetas a su responsabilidad. La ganancia en tiempo sería apreciable y la repetición del proceso de cálculo y simulación con un nuevo juego de datos, inmediata.

La aproximación gráfica del nuevo proceso comunicativo Alumno–Interfaz Independiente–MATLAB quedaría como se muestra en la Fig.3:

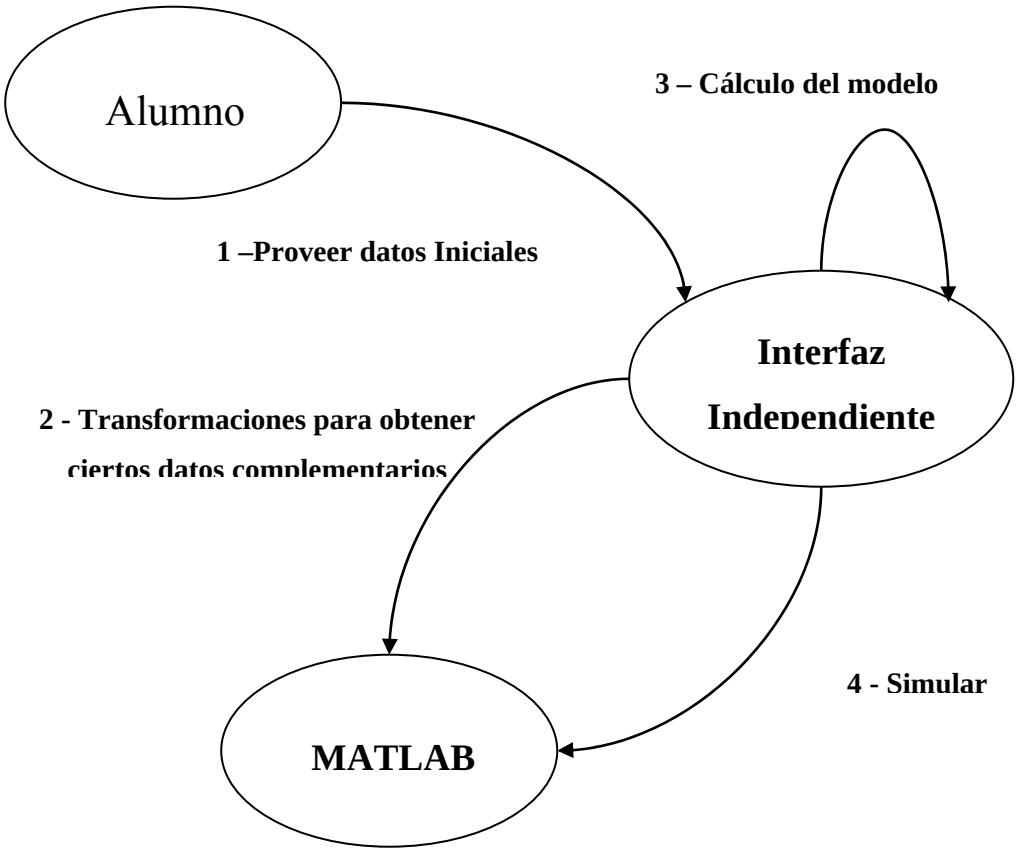

Fig.3 La única responsabilidad del alumno, según el método propuesto, es proveer los **datos iniciales**.

1.5.1 Estudio del estado del arte

La revisión de la bibliografía especializada, así como la búsqueda en Internet hechas para investigar qué trabajos se han realizado con propósitos similares a los nuestros, arrojan la confección de una gran cantidad de programas [5] [10] [11] que incluyen dentro de sus posibilidades, el cálculo de modelos de evaporadores para su posterior simulación, junto con los de otras plantas muy comunes en la industria (Columnas de destilación, Reactores, Generadores de vapor, Mezcladores, entre otros). Sin embargo son muy costosos. Existen sitios Web [9] que dan la posibilidad de calcular ciertos parámetros de evaporadores *On-Line* según datos proporcionados por el visitante, sin embargo, la simulación no es una opción contemplada dentro de los servicios. Los resultados que ofrecen son acerca de

parámetros muy generales que no brindan mucha información útil que pueda ser utilizada para verificar algún caso de estudio, al menos de mediana complejidad. El aprovechamiento de las potencialidades que ofrece MATLAB, el cual esta instalado en todas las computadoras utilizadas durante las horas clase dedicadas a las prácticas de simulación, hace de nuestra opción una variante muy útil que puede ser acogida sin gastos extras por las instituciones que decidan aplicarla.

1.5.2 Ventajas

La realización de esta herramienta y su inclusión dentro del apoyo de software para el diseño y análisis de evaporadores de calandria vertical en la disciplina de Sistemas de Control, fundamentalmente en la asignatura Procesos, será un paso de avance en pos de la simplificación de estos aspectos conllevando al mayor aprovechamiento docente de la actividad. El incremento de la autonomía conferida al estudiante en el desarrollo de esta tarea está a tono con la próxima aplicación del Plan de Estudios D en la educación superior, el cual concibe una disminución de la presencia del profesor en las actividades docentes aumentando las horas dedicadas a las actividades independientes.

1.6 Conclusiones

La complejidad de los sistemas multivariables, como es el caso de un evaporador, hace necesaria la utilización de MATLAB, como la mejor opción de software, para calcular sus modelos y simularlos. La gran variedad de usos que puede tener este programa lo obliga a tener una interfaz de usuario básica muy elemental. En pos de eliminar esta dificultad, para el caso del modelo de un evaporador, se decide la creación de una aplicación que aproveche las ventajas de MATLAB y a la vez garantice una mejor interacción con el usuario.

Capítulo 2

2. CONFECCION DE LA APLICACION

2.1. Introducción

En este capítulo se trata el proceso de desarrollo del software que resuelve las dificultades planteadas en el anterior. Se establecen con detalle los requerimientos que debe tener la solución para el problema. Se describen los métodos empleados para solucionar el aspecto de la comunicación MATLAB-Aplicación. Se comentan fragmentos del código del programa para ejemplificar aspectos claves del funcionamiento de este. Se describe el aspecto visual de la aplicación.

2.2 COM en nuestra aplicación.

Un pilar fundamental sobre el que se sustenta la concreción de nuestros objetivos, es precisamente el uso de una extensión de esta tecnología usada para resolver los aspectos comunicativos entre aplicaciones, llamada *OLE Automation*. En el curso de las explicaciones contenidas en este capítulo se hará uso reiterado de terminología y se referirán mecanismos propios de COM. Para entender más a fondo dichas especificidades destacaremos aspectos esenciales de esta fundación tecnológica. El apoyo bibliográfico para la confección de este epígrafe esta referenciado por [4], [13] y [18].

2.2.1 Consideraciones sobre la tecnología COM

COM (*Component Object Model*, según sus siglas en inglés) es un modelo binario propio de la plataforma Windows® para promover la interoperabilidad en el software, esto es, que permite la integración de aplicaciones o "componentes" dentro de un software con propósitos específicos, aún estando escritas en distintos lenguajes de programación, por distintos desarrolladores y en distintas épocas. Para soportar estas características de interoperabilidad, COM define e implementa mecanismos que permiten a las aplicaciones conectarse entre ellas como "Objetos de Software". Un "Objeto de Software" es un conjunto de capacidades que lo hacen funcional de algún modo así como el estado asociado a estas.

Un componente de software puede ser una estructura de datos, o un componente arquitectónico de software o un módulo de programa. En cada caso, ha de haber sido

diseñado de forma que facilite su reutilización sin necesidad de conocer los detalles de su funcionamiento interno. Un componente de software consta de uno o más objetos, donde cada objeto ofrece su funcionabilidad y contenido a través de una o más Interfaces, las cuales a su vez, poseen una o más funciones que realizan acciones específicas que constituyen los servicios que brinda el objeto.

2.2.1.1 COM como especificación e implementación

COM es una especificación y a la vez una forma de realización. La especificación de COM define cómo se comunican entre sí los objetos, a través del estándar binario de interfaces. Según esta especificación, los objetos COM pueden escribirse en diferentes lenguajes de programación y ser creados en procesos o maquinas diferentes. Siempre y cuando los objetos se ajusten a la especificación de COM, estos podrán comunicarse entre sí.

La implementación de COM se hace dentro del subsistema *Win32*, que proporciona varios servicios internos que apoyan su especificación. La biblioteca COM contiene, además de un conjunto de interfaces básicas que definen la funcionalidad esencial de un objeto COM, un grupo de funciones API (*Application Programming Interface*) que facilitan la creación y manipulación de dichos objetos, ya sean clientes o servidores. Esta biblioteca provee:

- Servicios localizadores de implementación, el cual a partir de un identificador de clase (CLSID) determina qué servidor implementa esa clase y dónde se encuentra, interponiendo entre la identidad de una clase de objeto y su implementación, un nivel de abstracción que es el registro del sistema para absorber futuros cambios en el código del servidor.
- Igualmente y mas importante aún, da la posibilidad de hacer uso de funcionalidades exportadas por un objeto COM creado en una computadora remota sobre la red o localmente, pero en otro proceso.
- Un mecanismo estándar para permitirle a una aplicación COM cualquiera controlar cuanta memoria está asignada dentro de sus procesos.

2.2.1.2 Interfaces COM

Los objetos COM se comunican entre sí a través de interfaces, estas son grupos de rutinas relacionadas lógica o semánticamente que proporcionan la comunicación entre el proveedor

de un servicio (Objeto Servidor) y sus clientes. Los objetos pueden tener múltiples interfaces, donde cada interfaz implementa una o varias funciones. Una interfaz proporciona una manera de comunicarle al cliente qué servicios esta proporciona, sin revelar detalles acerca de cómo o desde dónde el objeto proporciona este servicio.

Una vez publicadas, las interfaces son inmutables; es decir, no cambian. Se le puede asignar a una interfaz una funcionalidad específica en un momento dado. Si más adelante se quisieran aumentar los servicios que ofrece un objeto COM, es necesario adicionar otra interfaz.

Pueden ser redireccionadas por COM a través de *proxy's* para posibilitar llamadas a métodos de interfaces pertenecientes a objetos que estén corriendo en otros hilos, procesos, o máquinas remotas, siendo este proceso de comunicación completamente transparente para los objetos cliente y servidor.

2.2.1.2.1 Interfaz *IUnknown*

Todos los objetos COM deben implementar la interfaz fundamental, llamada *IUnknown*. Es una definición de tipo para la interfaz básica, la *IInterface*. *IUnknown* contiene las rutinas siguientes:

- *QueryInterface*: Proporciona los punteros a las otras interfaces que el objeto implementa. Y puede asignar un puntero hacia cualquier interfaz pedida por un cliente. Cuando este recibe el puntero hacia la interfaz solicitada, puede llamar a cualquier método perteneciente a dicha interfaz
- *AddRef* y *Release*: Métodos de conteo de simple referencia para determinar si el objeto COM esta siendo utilizado por clientes y el número de estos, de manera tal que se autodestruya cuando no haya ningún cliente requiriendo de sus servicios. De esta forma los objetos llevan cuenta de su propio período de vida liberando la memoria que ocupan cuando son desechados

2.2.1.3 Clientes COM

Cliente: La definición general es un fragmento de código que está usando los servicios de algún otro objeto, siempre y cuando este último pueda ser implementado.

Los clientes COM pueden consultar en cualquier momento qué interfaces implementa un servidor para determinar la funcionalidad que este es capaz de proporcionarle. Todos los servidores COM permiten a los clientes, pedidos a sus interfaces. Además, si el servidor soporta la interfaz *IDispatch*, los clientes pueden preguntar al servidor información sobre qué métodos soporta una interfaz determinada. Los clientes no necesitan saber cómo (o incluso dónde) un objeto proporciona los servicios que ofrece; ellos simplemente suponen que los objetos servidores proporcionan los servicios que anuncian a través de sus interfaces.

Hay dos tipos de clientes COM, Controladores y Contenedores. Los Controladores despiertan el servidor e interactúan con él a través de sus interfaces. Estos solicitan los servicios del objeto COM que despiertan o lo controlan como un proceso separado. Los Contenedores acogen controles visuales u objetos que aparecen en la interfaz de usuario del propio Contenedor. Este tipo de cliente usa interfaces predefinidas para negociar cuestiones visuales con los objetos servidores. Es imposible que exista una comunicación donde el cliente sea un Contenedor a través de DCOM, por ejemplo: Los controles visuales que aparecen en una interfaz de usuario del contenedor deben estar ubicados localmente. Esto es porque se espera que los controles se pinten por sí mismos, para lo que deben tener acceso a los recursos de GDI (*Grafic Devices Interface*) locales.

2.2.1.4 Servidores COM

Un servidor COM es una aplicación (.exe) o una biblioteca (.dll) que proporciona servicios a una o a varias aplicaciones o bibliotecas cliente. Un servidor consta de uno o más objetos COM que engloban cada uno un conjunto de propiedades y métodos.

Cuando un cliente quiere hacer uso de los servicios de un servidor COM, este llama la función API de la librería COM *CoCreateInstance* pasando el identificador de clase (CLSID) del objeto deseado como parámetro, la cual devuelve un puntero a la interfaz básica del objeto creado, también debe especificar qué tipo de servidor desea (In-process, Out-of-process o remoto). Un CLSID es simplemente un GUID (*Globally Unique ID*) que identifica una clase de objeto COM. COM usa la información guardada en el registro del sistema asociada a ese CLSID para localizar el modulo del servidor. Una vez localizado el servidor, COM carga el código en memoria, e instancia un objeto servidor de la clase

especificada por el cliente retornando un puntero a la interfaz básica de este. Este proceso es llevado a cabo por un objeto especial llamado *Class Factory*, el cual implementa la interfaz *IClassFactory*, cuyo propósito es el de crear los objetos de un CLSID particular. La implementación del objeto *Class Factory* pertenece al módulo del servidor que lo acoge. El servidor no es el objeto en sí, sino la estructura que, obedeciendo a las necesidades de los clientes, crea objetos de la clase especificada por ellos para servirles a través de las funcionalidades que ofrecen.

2.2.1.4.1 Servidores *In-process* y *Out-of-process*.

Con COM, un cliente no necesita saber donde reside el objeto al cual se quiere conectar, simplemente hace una llamada a la interfaz que quiere usar. COM realiza los pasos necesarios para hacer la llamada. Estos pasos difieren en dependencia de si el servidor reside en el mismo proceso que el cliente, en un proceso diferente, o en una máquina remota conectada en red. Los diferentes tipos de servidores son conocidos como:

- **Servidor *In-process*:** En este caso el código del servidor está contenido en una biblioteca (.dll) corriendo en el mismo espacio de direcciones del proceso del cliente. Este se comunica con el servidor *In-Process* haciendo llamadas directas a las interfaces del servidor.
- **Servidor *Out-of-process* (o servidor local):** Es el caso donde el código del servidor reside dentro de una aplicación (.exe) corriendo en un espacio de direcciones diferente, pero en la misma máquina que el cliente.
- **Servidor *Remote* (Remoto):** El servidor reside en una librería (.dll) o en un ejecutable (.exe) cargados en una máquina remota en la red.

2.2.1.5 Marshaling

Marshaling es el mecanismo que permite a un cliente realizar llamadas a la función de una interfaz de un objeto remoto que reside en otro proceso o en una máquina diferente. Ante la llamada a un método de una interfaz contenida en un objeto residente en otro proceso o computadora, *Marshaling* realiza los siguientes pasos.

- Toma un puntero a la interfaz en el proceso del servidor y lo hace disponible al código en el proceso del cliente a través de un puntero *proxy*.

- Transfiere los argumentos de una llamada a dicha interfaz hecha desde el cliente y ubica los argumentos en el espacio del proceso del objeto remoto para completar la llamada.

Para cualquier llamada a un método de una interfaz, el cliente pone los argumentos en una pila e invoca la función a través del puntero de la interfaz que la contiene. Si la llamada al objeto no es *In-Process*, esta se pasa al *proxy*. El *proxy* agrupa los argumentos en un paquete *Marshaling* y transmite la estructura al objeto remoto. Una subrutina (*stub*) en el objeto remoto desempaqueta los argumentos, los pone en una pila, y ejecuta la función del objeto. En esencia, el objeto recrea la llamada del cliente en su propio espacio de direcciones.

2.2.1.6 Las extensiones COM

A través del tiempo COM ha evolucionado y se ha extendido más allá de los servicios básicos que este ofrece. COM sirve como base para otras tecnologías como Automation, Controles *ActiveX*, *Active Documents*, *Active Directory*, COM+.

2.2.1.6.1 *Automation*

Automation refiere la habilidad de una aplicación para controlar otra de forma programática. El objeto servidor, que es manipulado, se llama Objeto *Automation*, y el cliente es denominado Controlador *Automation*. Esta extensión puede ser utilizada en servidores *In-process*, *Out-of-process* y Remotos.

El objeto *Automation* define un conjunto de propiedades y comandos, y describe sus capacidades a través de informaciones de tipo. Para hacer esto existe un modo de proveer información acerca de sus interfaces, los métodos de esas interfaces y los argumentos requeridos por dichos métodos. Normalmente esta información esta disponible en una Librería de Tipo (*Type Library*). El servidor *Automation* puede además generar, dinámicamente, información de tipo a través de la interfaz *IDispatch* que implementa.

2.2.2 Facilidades COM que ofrece MATLAB

MATLAB [17] soporta dos de las extensiones COM: Controles *ActiveX* y *Automation*. Las capacidades *Automation* incluyen la posibilidad de ejecutar comandos en el *workspace* e

insertar y extraer matrices de este. Precisamente estas son las características de manipulación de MATLAB que explotaremos para llevar a cabo nuestro proyecto. Hay que destacar que programativamente podemos fabricar clientes COM desde el entorno MATLAB haciendo uso de su lenguaje propio, con los que se podría manipular otros objetos COM que actuarían como servidores; pero nuestro objetivo aquí es explotar las capacidades como servidor *Automation* que este ofrece para poder despertar la aplicación MATLAB y controlarla desde otra desarrollada por nosotros. MATLAB también ofrece posibilidades de comunicación con otras aplicaciones a través de DDE (*Dynamic Data Exchange*), pero hemos escogido COM por las mayores facilidades que ofrece, porque es una tecnología que se ha hecho un estándar en la plataforma Windows® y porque es precisamente la alternativa superior creada por Microsoft® para los mismos efectos.

El nombre o ProgID que se asocia en el registro del sistema con el CLSID del servidor Automation de MATLAB es "Matlab.Application". Proporcionando este nombre a la hora de crear una instancia del servidor *Automation* desde una aplicación cliente, se despierta MATLAB y se le puede comenzar a controlar. La interfaz que ofrece su servidor *Automation* implementa tres métodos fundamentales que pueden ser invocados desde la aplicación con la que se pretende controlarlo, estos son: **Execute**, **GetFullMatrix** y **PutFullMatrix**. Además esta interfaz exporta también algunas funciones relacionadas con el comportamiento de la ventana de comandos. Estas son: **MinimizeCommandWindow**, **MaximizeCommandWindow** y **Visible.**

Execute acepta como parámetro una cadena que contiene el comando a ejecutar en el *workspace* de MATLAB. Utilizando **GetFullMatrix** podemos extraer una variable numérica (ya sea uni o bidimensional) del *workspace* pasando como parámetros, el nombre de dicha variable, el nombre del *workspace* donde se encuentra, y los arreglos donde se van a guardar la parte imaginaria y la parte real de la variable cuyo contenido se pretende transportar. **PutFullMatrix** es la operación inversa de **GetFullMatrix**, o sea, podemos ubicar variables, desde nuestra aplicación, en el *workspace* de MATLAB. En este caso se pasan los mismos parámetros: una con el nombre del *workspace* donde se va a ubicar, y el nombre que va tener la variable que contenga dicho valor y los arreglos correspondientes. Si la variable que se pretende crear solo tiene valores reales, el arreglo que contendría la parte imaginaria se pasa vacío.

Cuando el servidor *Automation* de MATLAB es creado de forma manual se le adiciona el comando "/Automation" en el proceso de inicialización de la aplicación. Microsoft® Windows® hace esto de manera automática cuando algún controlador lanza la aplicación. No obstante, si MATLAB ya esta corriendo y fue iniciado sin la línea de comando "/Automation" y algún controlador quiere establecer conexión con él a modo de cliente, esto provocara que Windows® cree otra instancia de MATLAB con el comando "/Automation". Esto protege de la posible interferencia que los controladores pudieran crear sobre cualquier otra sesión de MATLAB abierta previamente.

MATLAB es un servidor *Automation* de usos múltiple o dedicado, lo cual significa, en el caso de uso múltiple, que varios clientes pueden conectarse a una misma sesión de MATLAB lanzada como servidor *Automation*. Si alguna aplicación se inicia e intenta lanzar un servidor *Automation* de MATLAB de múltiple uso, Windows® chequeará si ya existe alguno corriendo, en caso negativo inicia uno, y en caso afirmativo conectará dicha aplicación irremediablemente al servidor que ya este activo. Si por el contrario, una aplicación lanza un servidor de uso dedicado con el ProgID "Matlab.Application.Single", se iniciara una sesión de MATLAB exclusiva para su uso. Destaquemos que todos los controladores conectados a un mismo servidor compartirán los mismos recursos de la sesión que estén usando, incluidas las variables y el *workspace*.

2.3 Desarrollo del Software

Las ventajas ofrecidas por la utilización del enfoque orientado a objeto en el desarrollo de software, se han hecho patente en su aplicación a casi todas las producciones de este tipo en la actualidad. Así mismo aplicamos este paradigma en la confección de la aplicación que nos ocupa. Sin ánimos de establecer fronteras definidas entre las etapas que definen el ciclo de vida del software según este enfoque, ni de redefinir los nombres que reciben los diferentes pasos aconsejados a realizar en cada una de estas [7], y solo con el propósito de simplificar la explicación de la metodología seguida, procedemos a la división del desarrollo de nuestra aplicación en:

2.3.1Análisis de Requisitos.

Según la descripción del problema realizada en el Capítulo 1, y apoyados en las entrevistas con los profesores que van a utilizar la aplicación en el desarrollo de sus clases, se obtuvo la especificación de los requisitos con los que debe cumplir el software a realizar:

Se necesita de una aplicación para facilitar el trabajo durante las labores de diseño y simulación de evaporadores de calandria vertical en las actividades docentes de las asignaturas de la disciplina de Sistemas de Control. Permitirá calcular el modelo en el espacio de estado y simular el comportamiento de la respuesta de cualquier par entrada-salida elegido por el alumno de manera rápida y cómoda para ellos a través de una interfaz de usuario "amigable". Mostrará de manera gráfica las matrices de estado calculadas. Debe liberar al alumno de la necesidad de la utilización de tablas para la búsqueda de datos requeridos en el cálculo del modelo. Debe incluir en los gráficos de las respuestas obtenidas, la representación de la función transferencia que relaciona el par de variables simulado. Tiene que hacer uso del programa MATLAB para aprovechar sus potencialidades de cálculo y graficación de datos. Alertará al alumno de forma visual y sonora sobre cualquier anomalía ocurrida durante el proceso de diseño y simulación, generando una lista de mensajes para su posterior revisión. Dará posibilidades de hacer persistente el juego de datos inicial provisto por el alumno para el cálculo, así como cargarlos desde disco sin necesidad de entrarlos por teclado en caso de que este así lo desee.

La necesaria cooperación entre nuestra aplicación y el programa MATLAB hace preciso definir el aspecto de la comunicación entre ambos. Las especificaciones hechas acerca de esta cuestión en el epígrafe 2.2.2 y, teniendo en cuenta que en Cuba el sistema operativo Windows® es el más ampliamente difundido, la comunicación que se establecerá con MATLAB se llevará a efecto haciendo uso del servidor *OLE Automation* que este implementa en su versión para el sistema operativo de Microsoft®. En el ambiente de programación Borland Delphi7, la creación de interfaces gráficas refinadas se hace con una facilidad extrema y se soporta la programación COM Estas características lo hacen nuestra elección para la elaboración de la aplicación.

2.3.2Casos de uso fundamentales de la aplicación

Basados en la descripción de la aplicación hecha en 2.3.1 establecemos las interacciones fundamentales que existirán entre el usuario (el alumno en este caso) y la aplicación. Las operaciones que se realizan tras los eventos generados por el usuario son mostrados en un diagrama de casos de uso. Empleando la notación *UML* (*Unified Modelling Language*, según sus siglas en inglés) [2] [14] [18], y con el apoyo de la herramienta CASE Rational Rose queda el diagrama como se muestra en la Fig.1.

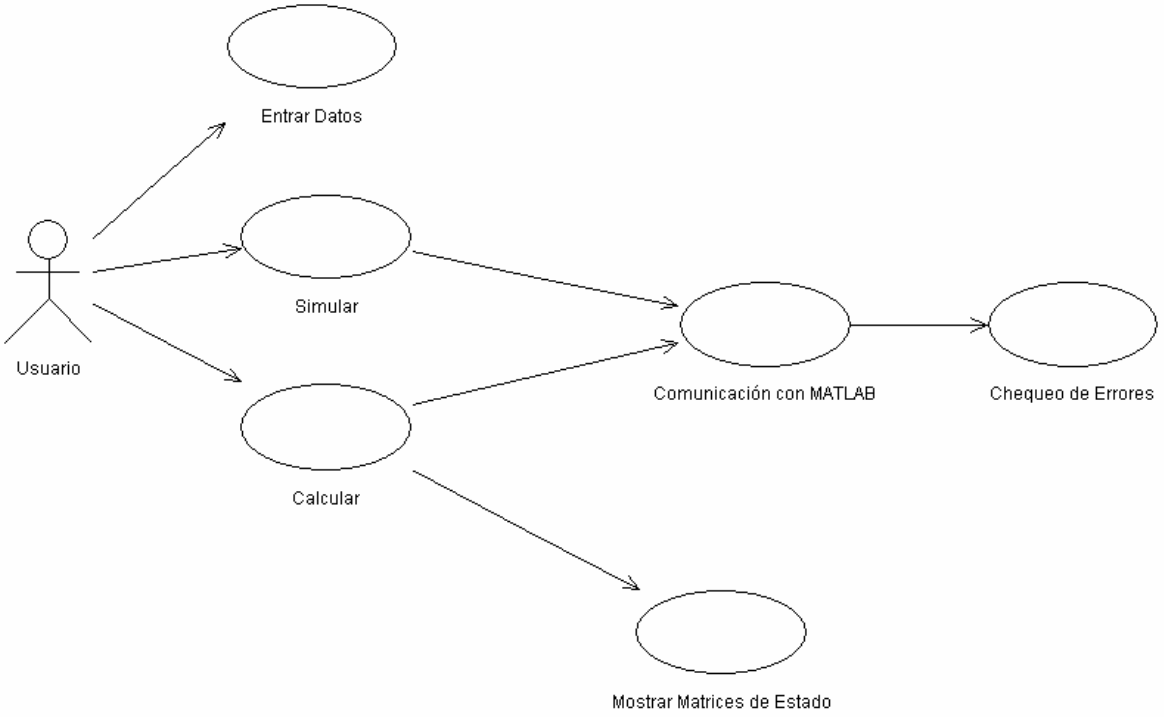

Fig.1: Diagrama de casos de uso que muestra la funcionalidad general de la aplicación.

2.3.3 Identificación de objetos.

Antes de proceder definamos qué es un objeto:

"Un objeto tiene estado, comportamiento e identidad; la estructura y el comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables" [3].

A través de la revisión minuciosa de las especificaciones de la aplicación identificamos las entidades siguientes.

- Manipulador Gráfico: Responsable de la manipulación de todos los eventos visuales, a través de los cuales el usuario interactúa con la aplicación.

- Evaporador: Contiene los atributos y operaciones que definen a una entidad como el modelo de un evaporador de calandria vertical dentro del contexto de la aplicación.
- Interfaz con MATLAB: Es la entidad encargada de la comunicación de la aplicación con MATLAB. Incluye la ejecución de las funciones que exporta el servidor *Automation* de MATLAB.
- Visualizador de Ecuaciones: Construye gráficamente la representación matricial clásica del modelo en el espacio de estado de cualquier sistema.

La identificación de las entidades hasta este momento se ha hecho siguiendo lo que sugiere la descripción del funcionamiento de la aplicación. Previendo alguna intención posterior de aumentar la cantidad de sistemas a simular en ella, creamos una entidad más abstracta que contenga rasgos y comportamientos comunes en todos los objetos que representen cualquier sistema que posea un modelo definido en el espacio de estado y que se desee simular por el usuario. Agregamos entonces a las entidades anteriores la que sigue:

- Modelo: Constituye un objeto abstracto que posee los atributos y realiza las operaciones que identifican el modelo en el espacio de estado de cualquier sistema dentro del contexto de la aplicación.

2.3.4 Diseño de la jerarquía de clases

Graficando el modelo de clases, apoyados en la herramienta CASE Rational Rose, según la notación UML, y considerando solamente los métodos y los campos fundamentales de cada clase se confeccionó el diagrama que se muestra en la Fig.2:

2.3.5 Implementación

Analicemos ahora con más detalle los aspectos estructurales de las clases *TModel*, *TEvaporatorModel* y *TMatlabInterface* (nombres según el código del programa). La comprensión del funcionamiento de estas entidades determina la posterior asimilación de los mecanismos implementados para el cumplimiento de los requisitos impuestos al inicio del desarrollo del software.

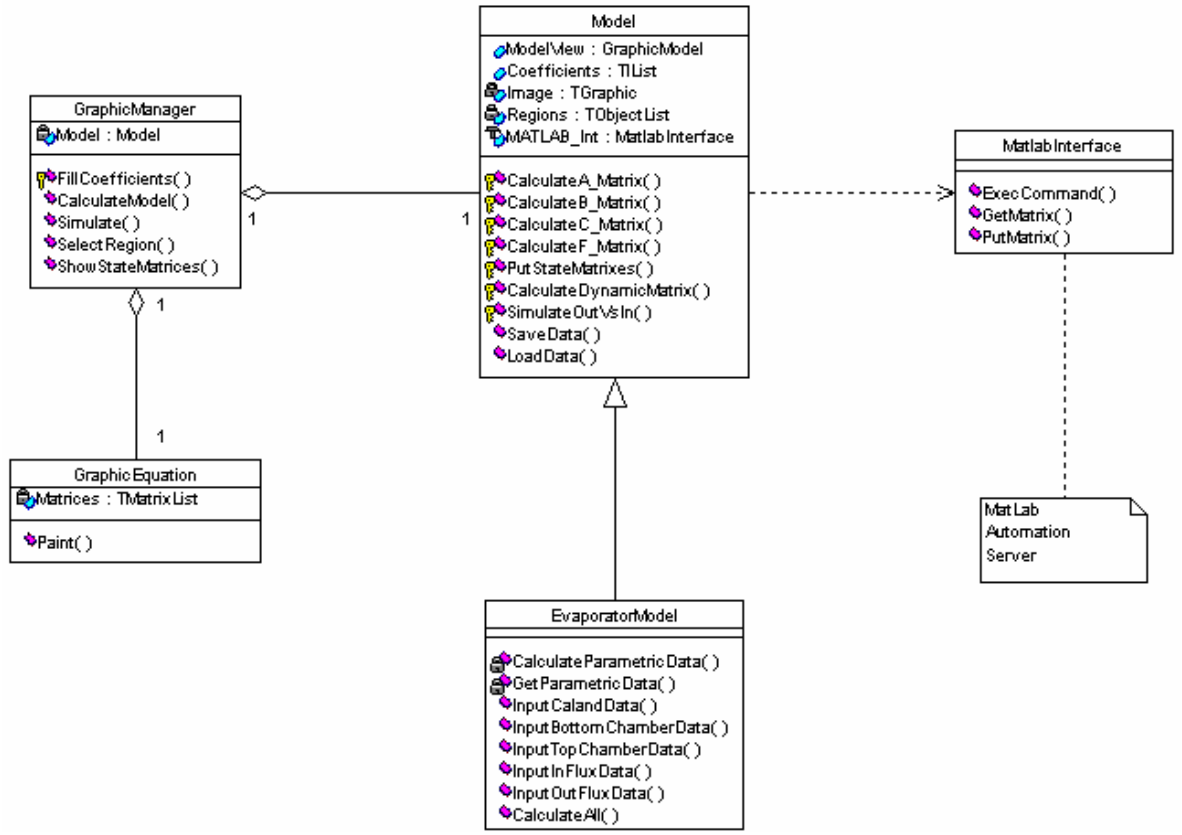

Fig.2: Diagrama de clases de la aplicación (notación UML).

2.3.5.1 La clase *TMatlabInterface*

Esta clase juega un rol fundamental dentro de la concepción funcional del producto que se desarrolla. Es el puente de comunicación entre la aplicación que interactúa con el usuario y el *engine* de cálculo de MATLAB. La comunicación entre aplicaciones como dijimos se hace a través de la tecnología COM, específicamente usando *OLE Automation*. Dicho esto, expliquemos las características fundamentales de la clase *TMatlabInterface*. A continuación mostramos su declaración:

```
TMatlabInterface =
    class
      private
        fMatlabCOMObject : variant;
        fMatlabWorkDir : string;
        fErrorsAtStart : TStringList;

        procedure SetVisibleMatlab(State : boolean);
        function InitilizeMatlab : boolean;
        function ReInitializeMatlab : boolean;
        function ExceptionHandler(ErrorCode : integer): byte;
```

```
public
   property VisibleMatlab : boolean write SetVisibleMatlab;
   property ErrorsAtStart : TStringList read fErrorsAtStart;
   function ExecCommand(CommandList: TStringList) : TExecuteErrorMess;
   function GetMatrix(MatrixExchangeArgList : TMatrixExchangeArgList) :
            TExecuteErrorMess;
   function PutMatrix(MatrixExchangeArgList : TMatrixExchangeArgList) :
            TExecuteErrorMess;
   constructor Create(CurrDir : string);
   destructor Destroy; override;
end;
```

El campo *fMatlabCOMObject* es la variable que guarda la referencia a la interfaz *IDispatch* del servidor *Automation* de MATLAB cuando éste es lanzado desde nuestra aplicación al iniciarse. O sea:

```
fMatlabCOMObject := CreateOleObject('Matlab.Application');   // Se instancia el servidor
                                                              //Automation
```

La propiedad ***VisibleMatlab*** determina si la ventana de comandos de la sesión de MATLAB abierta al crear el servidor *Automation* será visible o no.

ExecCommand, ***GetMatrix***, ***PutMatrix*** son funciones que engloban dentro de sí las funciones que exporta MATLAB (**Execute**, **GetFullMatrix**, **PutFullMatrix**) a través de la interfaz COM que las contiene. La declaración de dichas funciones obtenidas de la información de tipo que provee la *Type Library* del objeto *OLE Automation* que exporta MATLAB son las siguientes (en sintaxis de Delphi):

```
function Execute (const Command: WideString): WideString; stdcall;
function GetFullMatrix (const Name: WideString; const Workspace: WideString;
                        var pr: PSafeArray; var  pi: PSafeArray): HResult; stdcall;
function PutFullMatrix (const Name: WideString; const Workspace: WideString; pr: PSafeArray;
                        pi:  PSafeArray): HResult; stdcall;
```

El argumento que acepta cada una de estas funciones es único, es decir, si se quiere ejecutar una secuencia de comandos habría que invocar la función *Execute* tantas veces como comandos tuviésemos que ejecutar. Igualmente ocurre con el resto de las funciones. En caso de que necesitáramos transportar varias matrices en un sentido u otro de la comunicación entre las dos aplicaciones, habría que invocar las funciones al efecto tantas veces como el número de matrices a transportar.

En nuestro caso particular necesitamos ejecutar, en ciertos puntos del programa, secuencias de comandos en MATLAB, así como transportar (varias de una vez) matrices con datos necesarios para el desenvolvimiento de los cálculos y la simulación. Obviamente la eficiencia de las funciones originales de MATLAB no es la mejor en el contexto de nuestro problema específico, por eso decidimos crear unas nuevas, basadas en ellas, que se adaptaran mejor a nuestras necesidades. Implementamos las funciones ***ExecCommand***, ***GetMatrix*** y ***PutMatrix*** para, en lugar de un argumento único, pasarles como parámetro un arreglo de estos, o sea, si tenemos una lista de comandos a ser ejecutados en MATLAB, simplemente se invoca nuestra función ***ExecCommand*** una sola vez, pasando como argumento dicha lista y ella, por sí misma, se encarga de llamar la función **Execute** original de la interfaz del servidor *Automation* pasándole cada comando de la lista uno por vez. Lo mismo ocurre con las funciones dedicadas al transporte de matrices.

Estas tres funciones devuelven al terminar el ciclo de ejecución una lista con los errores que pudieran ocurrir. La declaración de tipo del resultado de dichas funciones es como sigue:

```
type
  TErrorMess =
     packed record
       ErrorList : TStringList;
       CommandExecLogs : byte;
     end;
```

El campo *ErrorList* contiene todos los mensajes de error que devuelve MATLAB cuando se intentan realizar en él la lista de acciones pasada como parámetro. Y *CommandExecLogs* informa si la secuencia de acciones se culminó con éxito (valor 2) o si se vio interrumpida por la no disponibilidad del servidor *Automation* y se procede a la creación de una nueva sesión de MATLAB (1 si se reinicia con éxito y 0 si ocurre lo contrario).

Si por alguna equivocación o mala manipulación del usuario se cerrase la sesión de MATLAB a la cual está conectada nuestra aplicación, y más tarde se intenta realizar alguna operación con ella, o de la misma forma, ocurre durante un ciclo de ejecución dentro de cualquiera de las funciones ***ExecCommand***, ***GetMatrix*** o ***PutMatrix***, inmediatamente se le alerta del suceso y se le pregunta al usuario si desea volver a despertar MATLAB. En caso afirmativo se libera la variable *fMatlabCOMObject* y se procede a crear una nueva instancia del servidor *Automation*. Existe una serie de errores contemplados en la operación

de inicio de MATLAB que le informan al usuario qué ocurre en caso de que no pueda despertar.

Cada vez que MATLAB es lanzado con éxito, se ejecutan de manera automática una serie de comandos (utilizando la función ***ExecCommand***) que lo preparan para servir en las tareas del cálculo de los datos complementarios. Esto es, que de primera instancia, el directorio de trabajo de MATLAB se sitúa en el mismo donde reside la aplicación. Se realiza esta operación porque adjunto al ejecutable del programa esta un fichero llamado "TablasEvaporador.m" (obligatoriamente dentro de una carpeta llamada "Tablas") donde se declaran los vectores con los valores de las entradas de las tablas necesarias para buscar los datos complementarios obligatorios para el cálculo del modelo. Acto seguido se le ordena a MATLAB ejecutar ese fichero para cargar en el *workspace* los vectores allí declarados. Luego se ajustan las curvas que relacionan los pares de vectores que contiene el *.m* utilizando el comando de MATLAB *polyfit* y se hallan las derivadas de sus funciones mediante la combinación de *sym2poly* y *diff*. La propiedad *ErrorsAtStart* lee del campo privado *fErrorsAtStart* y contiene los errores producidos durante la ejecución de esta secuencia inicial de comandos.

2.3.5.2 La clase *TModel*

Esta clase es el ancestro común de todas las clases que especifiquen el modo de hallar el modelo en el espacio de estado de cualquier sistema que se pretenda calcular y simular en nuestra aplicación. Su creación responde al nivel de abstracción necesario para admitir futuras ampliaciones del programa, donde no solo se contemple la posibilidad de calcular y simular evaporadores de calandria vertical. Aquí se incluyen los métodos y atributos comunes que estarán presentes en cada clase que vaya a implementar de manera concreta la obtención del modelo en el espacio de estado de algún dispositivo (ya sea un reactor, una columna de destilación, un generador de vapor, etc.) y su simulación. Veamos la declaración de esta clase.

```
TModel =
  class
    private
      fName : string;
      fPostMan : TMatlabInterface;
      fModelData : array of real;
```

```
    fA_Matrix : TMatrix;
    fB_Matrix : TMatrix;
    fC_Matrix : TMatrix;
    fF_Matrix : TMatrix;
    fErrorsCalculatingModel : TStringlist;

    function GetA_Matrix : TStringMatrix;
    function GetB_Matrix : TStringMatrix;
    function GetC_Matrix : TStringMatrix;
    function GetF_Matrix : TStringMatrix;
    function PutStateMatrixes : TExecutionStepSt;
    function CalculateDynamicMatrix : TExecutionStepSt;
    procedure CalculateA_Matrix; virtual; abstract;
    procedure CalculateB_Matrix; virtual; abstract;
    procedure CalculateC_Matrix; virtual; abstract;
    procedure CalculateF_Matrix; virtual; abstract;
  public
    property A_Matrix : TStringMatrix read GetA_Matrix;
    property B_Matrix : TStringMatrix read GetB_Matrix;
    property C_Matrix : TStringMatrix read GetC_Matrix;
    property F_Matrix : TStringMatrix read GetF_Matrix;
    property Errors : TStringList read fErrorsCalculatingModel;
    function SimulateOutVsIn(OutVar, Invar, VectorType, MultipleFigures : byte; OutName,
                                    InName: string;) : boolean;
    procedure SaveData(FileName : string);
    procedure LoadData(FileName : string);
    function CalculateAll : boolean; virtual; abstract;
    constructor Create(PostMan : TMatlabInterface); virtual;
  end;
```

El campo privado *fModelData*, que es un arreglo dinámico de valores reales, es la estructura que contiene los datos (**datos iniciales** y **datos complementarios** según el convenio hecho en la sección 1.4 del Capítulo 1) para calcular íntegramente los coeficientes de las matrices de estado que describen el modelo. En el método C*reate* de cada clase en particular que herede de *TModel* se la asignará un tamaño a *fModelData* de acuerdo con la cantidad de datos que se requieran para calcular los parámetros del modelo específico que implementen dichas clases. Las propiedades *A_Matrix*, *B_Matrix*, *C_Matrix* y *F_Matrix* contienen cada una, una matriz. Este número de matrices es el requerido para poder describir el modelo en el espacio de estado de cualquier sistema. El tamaño de estas estructuras de datos es dinámico y puede acoger el número de coeficientes que emplee un sistema en particular para ser descrito. La propiedad *Errors,* que lee del campo privado *fErrors,* contiene los errores ocurridos durante los procesos de cálculo y simulación. Todas

las clases que definan un modelo deben tener esta propiedad para informar sobre las anomalías que le impiden al usuario lograr sus objetivos con el programa.

Los procedimientos ***CalculateA_Matrix***, ***CalculateB_Matrix***, ***CalculateC_Matrix*** y ***CalculateF_Matrix*** estarán presentes en cualquier descendiente de esta clase pues necesariamente habrá que calcular en cualquier modelo las matrices que lo describen en el espacio de estado. Se declaran en este nivel de abstracción *virtual* y *abstract* porque se desconoce cual será su implementación futura, ya que esto depende de las características específicas del dispositivo representado por la clase hija de *TModel* y de su modelo matemático.

Todas las clases que descienden de *TModel*, implementarán los métodos ***PutStateMatrixes*** y ***CalculateDynamicMatrix*** ya que ejecutan las acciones relacionadas con los procesos de ubicar en el *workspace* de MATLAB las matrices de estado y hallar la matriz dinámica del sistema a partir de ellas. Detallemos aquí un fragmento del método ***PutStateMatrixes*** para observar cómo se realiza la configuración de una lista de matrices que se quiere transportar hacia MATLAB y la llamada a la función ***PutMatrix,***

```
function TModel.PutStateMatrixes : TExecutionStepSt;
 var
   CommandListToPutStateMatrixes : TMatrixExchangeArgList;
 begin
     ...
   Set Length(CommandListToPutStateMatrixes, 4);
   CommandListToPutStateMatrixes[0].Container := fA_Matrix;
   CommandListToPutStateMatrixes[0].VarNameInMatlab := 'A_Matrix';
   CommandListToPutStateMatrixes[1].Container := fB_Matrix;
   CommandListToPutStateMatrixes[1].VarNameInMatlab := 'B_Matrix';
   CommandListToPutStateMatrixes[2].Container := fF_Matrix;
   CommandListToPutStateMatrixes[2].VarNameInMatlab := 'F_Matrix';
   CommandListToPutStateMatrixes[3].Container := fC_Matrix;
   CommandListToPutStateMatrixes[3].VarNameInMatlab := 'C_Matrix';

   with fPostMan.PutMatrix(CommandListToPutStateMatrixes) do
     begin
      …
      …
      end;
```

La declaración del tipo *TMatrixExchangeArgList* se realiza en el módulo donde se implementa la clase *TMatlabInterface* y es la siguiente:

```
type
  TMatrixExchangeArg =
    Packed record
      VarNameInMatlab : String;
       Container : TMatrix;
end;
TMatrixExchangeArgList = array of TMatrixExchangeArg;
```

El método ***SimulateOutVSIn*** acepta como parámetros un par entrada-salida descrito en el modelo, para ser simulada su respuesta ante una entrada paso, la especificación de si el vector al que pertenece la entrada del par es el de control o el de las perturbaciones (según el modelo) y una variable booleana para decidir si el gráfico con la respuesta es pintado en una ventana propia o si sobrescribe la que ya está abierta. Aquí se construyen los comandos a ejecutarse en MATLAB para excitar con una entrada paso el par entrada-salida definido por las variables pasadas como argumentos a este método. Este proceso es, concretamente, excitar un elemento de la matriz dinámica del sistema que, como habíamos dicho, es la función transferencia que relaciona una entrada con una salida. Es decir, si la matriz dinámica de un sistema esta definida en el *workspace* de MATLAB como la variable *SysTFInControlVect*, entonces el comando "step (SysTFInControlVect (1, 3))" simulará la respuesta de la función transferencia, que relaciona la salida 1 con la entrada 3, ante una entrada paso.

Véase el fragmento de código que realiza la operación de construir el comando correcto para simular la respuesta de una de las salidas del modelo ante una excitación paso en una de sus entradas.

```
function TModel.SimulateOutVsIn (OutVar, Invar, VectorType, MultipleFigures : byte; OutName,
                                  InName : string) : boolean;  // VectorType 0 simular las
                                                               // salidas ante cambios en el
                                                               //vector de control, 1 el de las
                                                               //  perturbaciones.
 var
   TempList : TStringList;
   Caption : string;
 begin
    Result := true;
    TempList := TStringList.Create;
    Caption := ""+ OutName + ' vs. ' + InName + ' - Entrada Paso"';
    case VectorType of
      0:  TempList.Append('tf2text(SysTFInControlVect(' + inttostr(OutVar) + ', ' + inttostr(InVar) +
```

```
                                ), ' + inttostr(MultipleFigures) + ', ' + Caption + ')');
    1:  TempList.Append('tf2text(SysTFInPerturbationsVect(' + inttostr(OutVar) + ', ' +
                                 inttostr(InVar)+ '), ' + inttostr(MultipleFigures) + ', ' + Caption + ')');
  end;
  with fPostMan.ExecCommand(TempList) do
    begin
     if (ErrorList.Count <> 0)
      then
        begin
          ErrorList.Insert(0, '    ----Errores Simulando Modelo en Matlab ----');
          fErrors.Clear;
          fErrors := ErrorList;
          Beep;
        end;
    end;
   Result := false;
end;
```

Es necesario hacer notar que *tf2tex* es una función desarrollada en MATLAB para ser ejecutada como comando. Esta rutina es la encargada de construir la ventana con el gráfico de la simulación del par entrada-salida agregándole a esta los detalles informativos sobre la función transferencia que relaciona las variables del par, así como un texto en el nombre de la ventana identificándolas. El fichero "tf2tex.m" que contiene el código de esta función debe estar incorporado a la carpeta "Tablas" donde se alojan los ficheros con las tablas de parámetros.

El procedimiento ***SaveData*** es el encargado de hacer persistentes los **datos iniciales** que son parte del arreglo de datos que tiene como campo privado esta clase y por tanto cualquier descendiente de ella. Y ***LoadData*** lee de disco los datos guardados en él y los inserta en su lugar dentro del campo *fModelData*.

2.3.5.3 La clase *TEvaporatorModel*

Constituye una especialización de la clase *TModel* y especifica, según las características del dispositivo que representa (un evaporador de calandria vertical), los métodos declarados virtuales en aquella y agrega algunos más que son propios de esta clase, además de algunos atributos. La declaración de esta clase es:

```
  TEvaporatorModel =
    class(TModel)
      private
       …
       …
```

```
    fCommandListToParamData : TStringList;

    function CalculateParametricData : TExecutionStepSt;
    function GetParametricData : TExecutionStepSt;
    procedure CalculateA_Matrix; override;
    procedure CalculateB_Matrix; override;
    procedure CalculateC_Matrix; override;
    procedure CalculateF_Matrix; override;

  public
    procedure InputCalandData(St, U, A, Meo, P2o, Mso : string);
    procedure InputBottomChamberData(P1o, Glo, Vo, Vc, Pl, ho : string);
    procedure InputTopChamberData(Pco, Vv : string);
    procedure InputInFluxData(M1o, C1o : string);
    procedure InputOutFluxData(M2o, C2o : string);
    function CalculateAll : boolean;
    constructor Create(PostMan : TMatlabInterface); override;
  end;
```

Los métodos ***CalculateA_Matrix***, ***CalculateB_Matrix***, ***CalculateC_Matrix*** y ***CalculateF_Matrix*** tienen la implementación adecuada para obtener los coeficientes de las matrices que representan el modelo en el espacio de estado de un evaporador de calandria vertical. Las expresiones para calcularlos fueron tomadas de la literatura [1] [6].

CalculateParametricData es el método encargado de confeccionar la lista de comandos que van a evaluar las funciones creadas cada vez que inicia MATLAB, en los puntos de operación establecidos por el juego de datos proporcionado por el usuario, el resultado de dichas evaluaciones son los **datos complementarios** para el cálculo del modelo. El siguiente fragmento de código muestra unos pocos pasos para la confección de la lista que se guarda en el campo *fCommandListToParamData*.

```
function TEvaporatorModel.CalculateParametricData : TExecutionStepSt;
 var
   Command : string;
begin
   Result.Success := false;
   Result.StepErrors := TStringList.Create;
   fCommandListToParamData.Clear;
   Command := 'Ro1o = polyval(DenLiquidVSConcLiquid, ' + FloatToStr(fModelData[27]) + ')';
   fCommandListToParamData.Append(Command);
   Command := 'd1 = polyval(DenLiquidVSConcLiquid_Der,' + FloatToStr(fModelData[27]) + ')';
   fCommandListToParamData.Append(Command);
   Command := 'Ro2o = polyval(DenLiquidVSConcLiquid, ' + FloatToStr(fModelData[29]) + ')';
```

```
    fCommandListToParamData.Append(Command);
      …
      …
    with fPostMan.ExecCommand(fCommandListToParamData) do
      begin
        …
        …
      end;
end;
```

GetParametricData transporta los resultados que arrojó la ejecución de la lista de comandos confeccionada en ***CalculateParametricData***, desde MATLAB hacia nuestra aplicación, y los incorpora en el campo *fModelData* quedando todo listo para proceder a la sustitución de todos los parámetros en las expresiones que calculan los coeficientes de las matrices de estado. Los procedimientos ***InputCalandData***, ***InputBottomChamberData***, ***InputTopChamberData***, ***InputInFluxData*** y ***InputOutFluxData*** son llamados para incorporar los **datos iniciales** en el campo *fModelData* cuando estos son ingresados por teclado. Para facilitarle al usuario la asociación de estos datos con su representación física se han dividido según su relación con las partes fundamentales en las que descompusimos un evaporador: Calandria, Cámara Superior, Cámara Inferior, Alimentación y Salida de Líquido Evaporado.

2.4 Aspecto de la aplicación

Establecidos ya los mecanismos fundamentales empleados para el funcionamiento interno del programa, nos queda entonces describir cómo es el proceso de intercambio de información visual con el usuario. Este detalle es de suma importancia para cumplir con las exigencias planteadas en la etapa de análisis de los requisitos. El objetivo fundamental en esta fase de trabajo es lograr una interfaz en la que el usuario, con el menor esfuerzo posible, sea capaz de comunicarle al programa lo que desea hacer con él.

Utilizando los componentes de la VCL (*Visual Component Library*) de Delphi, los cuales recrean todos los controles visuales de Windows®, conformamos el ambiente gráfico que se precisa para, según las especificaciones hechas en la etapa de análisis, realizar de manera ágil y eficaz los procesos de entrada de los datos, cálculo y simulación del modelo de un evaporador, y la notificación de errores ocurridos durante todas las operaciones ejecutadas.

Fue necesario adicionar funcionalidades extras a algunos de estos controles visuales para ajustarlos mejor a nuestras necesidades.

La configuración de este ambiente gráfico es (usando la nomenclatura de la VCL):

- Una barra menú con tres submenús.
- Un *ListBox* para mostrar los errores ocurridos, acompañado de un *Image*
- Un *PageControl* con dos *TabSheet* los cuales contienen el grueso de los componentes visuales para realizar las operaciones deseadas.

2.4.1 Barra de menú principal

Contiene las operaciones relacionadas con:

- Los ficheros y el fin de la aplicación (Archivo).
- Los aspectos concernientes a las condiciones de trabajo con la aplicación (Herramientas).
- La Ayuda del programa, y la información sobre el nombre, los realizadores y la fecha de confección de la aplicación (Acerca de).

2.4.1.1 Menú Archivo

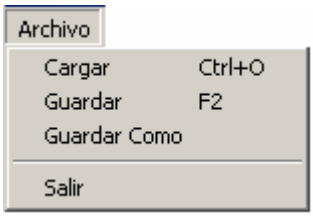

Fig.3: Menú "Archivo"

La opción "Cargar" le permite al usuario buscar el fichero con los datos previamente salvados de una sesión de trabajo anterior. Cuando se elige el fichero se llama el procedimiento ***LoadData*** de una instancia de la clase *TEvaporator* creada al inicio de la aplicación. Los ficheros que espera cargar la aplicación son de tipo texto y contiene las cifras entradas como datos en el orden según aparecen a la hora de ser entrados de manera visual. Antes de hacer la llamada a este procedimiento se verifica primero si el fichero contiene caracteres válidos que se puedan convertir a flotantes y verifica que solo tenga el número de líneas correspondientes al número de **datos iniciales**. Al cargar un juego de

datos, estos se ubican en los *Edits* correspondientes donde debieron ser escritos si se hubieran entrado por teclado.

La opción "Guardar" permite al usuario hacer persistente la lista de parámetros que entró, para si en una ocasión posterior desea analizar un sistema con los mismos datos, no tener necesidad de escribirlos nuevamente.

2.4.1.2 Menú Herramientas

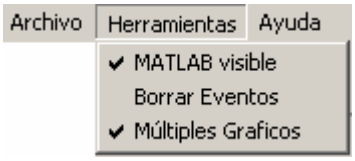

Fig.4: Menú "Herramientas"

- "MATLAB Visible" le permite al usuario ocultar la ventana de MATLAB o hacerla visible según esté marcada esta opción.
- "Borrar Eventos" borra el contenido del *StringList* que contiene la suma de todos los reportes de error que se han generado desde que se comenzó a utilizar la aplicación.
- "Múltiples Gráficos" determina si los gráficos con las simulaciones del par entrada-salida del sistema, escogido por el usuario se muestran en una misma ventana (sobrescribiendo el anterior) o en múltiples ventanas. Esta opción es útil en caso de que se quieran comparar varios gráficos.

2.4.1.3 Menú Ayuda

Fig.5: Menú "Ayuda"

- "Ayuda" muestra los tópicos que describen el funcionamiento de la aplicación y aconseja cómo tratar algunos errores típicos que pueden a parecer durante el uso de la aplicación.
- "Acerca de" Informa sobre la fecha de confección, la versión, el nombre y el autor de la aplicación.

2.4.2 Visualizador de eventos de error

La lista con todos los eventos de error ocurridos y la hora a la cual se produjeron es mostrada mediante un *ListBox*. Cada grupo de errores está encabezado por la identificación de la etapa de ejecución donde se detectaron.

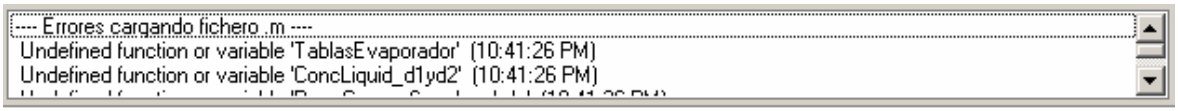

Fig.6: Lista de eventos de error

Próxima a ésta se encuentra una señalización visual que indica el estado de operación de la aplicación. Cuando ocurre un suceso de error, esta señal (un *bitmap* sobre la forma) cambia su aspecto llamando la atención del usuario para que inspeccione la lista de eventos.

2.4.3 Hojas de trabajo

Esta parte de la aplicación es un control visual llamado *PageControl*, el cual consta de dos *TabSheet* (llamados hojas).

2.4.3.1 Hoja de Cálculo

La primera hoja contiene el esquema de un evaporador de calandria vertical y un botón para confirmar que todos los datos están en su lugar y proceder a calcular el modelo. Deslizando el puntero del *mouse* sobre el esquema se descubrirá un mensaje (*hint*) informándole al usuario que puede hacer *click* en esa zona para entrar los datos correspondientes a la parte estructural del evaporador sobre la que está. Al hacer *click* se abre una ventana con la cantidad de *Edits* correspondiente al número de datos asociados, física o funcionalmente, con esa parte del evaporador. Estos *Edits* son un componente que hereda de *TCustomEdit*. Fue desarrollado para que solo aceptara caracteres válidos (números y punto) con el objetivo de no tener que validar en el código de nuestro programa, la cadena que entra el usuario por teclado.

2.4.3.2 Hoja de Simulación

La segunda hoja contiene un área para mostrar las matrices de estado calculadas a partir de los datos entrados en la hoja anterior. Estas matrices se muestran en la forma:

$$\dot{x} = Ax + Bu + Fv$$

$$y = Cx$$

Tiene además dos controles visuales que usamos para, con un *click* del *mouse*, escoger un par entrada-salida y automáticamente ordenar la visualización del gráfico con la respuesta de este ante una entrada paso. Estos controles son del tipo *TVariableLinkToSim*. Este es un componente desarrollado expresamente para esta aplicación que hereda de *TStringGrid* el cual es un control para mostrar datos tabulados en casillas. Esta característica se puede aprovechar para aproximar de cierta forma la ubicación de cada casilla en un plano coordenado donde cada una es un punto. Al deslizar el *mouse* sobre alguna casilla de este control se destacaran las que corresponden a su abscisa y a su ordenada. Convenientemente se ubican los vectores de salida en las abscisas y los de entrada en las ordenadas. Supongamos entonces, que, el usuario quisiese simular y graficar la respuesta de P_2 ante una entrada paso en S_0 tendría que buscar "P_2" en la fila superior del control y "S_0" en la columna izquierda. O sea:

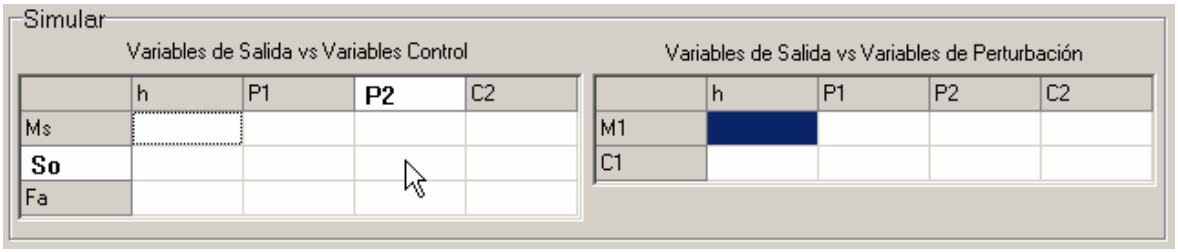

Fig.7: Sección para elegir el par entrada-salida a ser simulado

2.5 Conclusiones

Queda así conformada nuestra aplicación. Obsérvese que la entrada de los datos para dar inicio al proceso de cálculo del modelo, lo cual era un aspecto objetable en el método convencional utilizado, es muy intuitiva. El usuario puede guiarse por sí solo para insertarlos en el programa, al hacer la asociación de éstos con la parte estructural de un evaporador a la que pertenecen. Dedicado este software fundamentalmente a la docencia, el

alumno entiende mejor el funcionamiento de esta clase de dispositivos. Los procesos de cálculo y simulación del modelo se convierten ahora en meras operaciones de hacer *click* en los lugares indicados y escribir los datos en las ventanas correspondientes. La necesidad de poseer las tablas para obtener los valores de varios parámetros necesarios en el cálculo, queda eliminada por completo, pues solo hay que realizar el esfuerzo de transcribir el fragmento necesario para los casos de estudio proyectados, una sola vez. En caso de realizar una simulación utilizando un juego de datos ya utilizado y guardado en disco previamente, el proceso se agiliza aún más. El tiempo que media entre el conocimiento de un juego de datos para el cálculo y la simulación de un modelo y el análisis del gráfico con la respuesta es realmente pequeño. Se aprovechan todas las prestaciones del programa MATLAB y se lo dota, para este caso específico, de una interfaz de usuario completamente interactiva.

Capítulo 3

3. ANALISIS DE LOS RESULTADOS

3.1 Introducción

En este capítulo se comprueba en la práctica la eficacia del software desarrollado, tomando como ejemplo uno de los casos de estudio que se emplean en la realización de la tarea extraclase que sobre el tema de evaporación se orienta en la asignatura Procesos. Se hace una comparación entre las ventajas de la inclusión de nuestra aplicación en la enseñanza de las asignaturas de la disciplina de Sistemas de Control, sobre el método tradicional usado hasta ahora.

3.2 Cálculo y Simulación a partir de datos reales de un caso.

Dado un evaporador de calandria vertical, empleado para concentrar jugo claro en la industria azucarera, calcular su modelo en el espacio de estado y simular las respuestas de las salidas h, P_1, P_2 y C_2 ante variaciones en las señales de entrada correspondientes al mando: M_s, S_0, F_a y a las perturbaciones: M_1 y C_1.

Los datos de operación, son tomados de trabajos realizados anteriormente y se muestran a continuación:

Flujo de Entrada: 42.6931 m^3/h.
Flujo de Salida: 34.8508 m^3/h.
Flujo de Vapor a la Entrada de la Calandria: 8822.3716 kg/m^3.
Flujo de Vapor a la Salida de la Calandria: 7847.148 kg/m^3.
Concentración del jugo a la Entrada (C_1): 0.16.
Concentración del jugo a la Salida (C_2): 0.193.
Presión en la Cámara Sup. (P_1): 0.125 *Mpa.*
Presión en la Calandria (P_2): 0.164 *Mpa.*
Presión de Descarga: 0.037527 *Mpa.*
Altura del Líquido en la Calandria: 0.83 *m*
Sección Transversal total de los Tubos: 2.4744 m^2.
Volúmen del espacio situado Debajo de los Tubos: 4.2194 m^3.
Volúmen de la Cámara Sup.: 35.325 m^3.
Volúmen Exterior de los Tubos en la Cámara de Calefacción: 11.4765 m^3.
Cantidad de Líquido en el Evaporador: 6752.3470 *kg.*
Coeficiente Total de Transf. de Calor: 2000 $Kcal./h^0C$

Superficie Total de Transf. de Calor: 2.47 m^2.
Presión en el condensador (P_{co}): 0.088 *Mpa.*

Este es el juego de datos inicial para realizar las operaciones especificadas en la orden del ejercicio. En realidad los datos totales a sustituir en las expresiones del modelo son 31, de los cuales solo 18 son entregados. Utilizaremos para la resolución de este ejercicio la aplicación desarrollada en este trabajo y así comprobar su efectividad.

Una vez cargada la aplicación, el alumno tiene ante sí, una ventana donde se observa el esquema de un evaporador de calandria vertical. Una nota explicativa le sugiere recorrer con el puntero del *mouse* la figura para descubrir dónde hacer *click* y abrir las ventanas de entrada de datos. A medida que el alumno siga la indicación que se le sugiere y deslice el puntero del *mouse* sobre el esquema, irán apareciendo, según la representación de la parte estructural del evaporador sobre la que esté situado, textos (*hints*) advirtiéndole que puede hacer *click* y abrir la ventana para la entrada de los datos correspondientes a esa parte. La Fig.1 muestra todos los posibles mensajes.

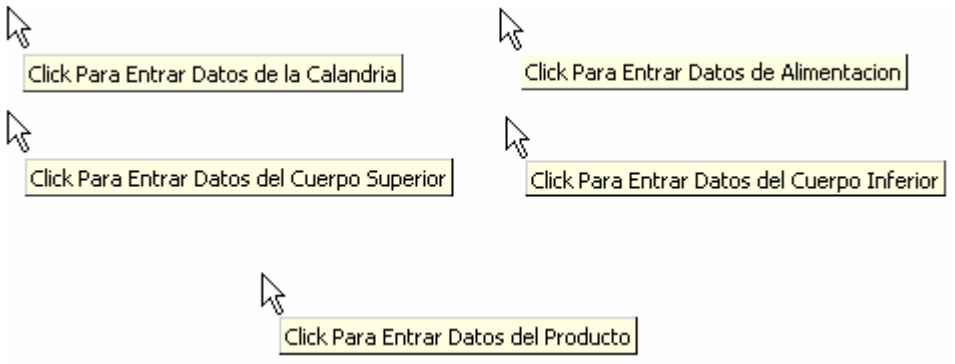

Fig.1: "*Hints*" con los mensajes para abrir las ventanas de los datos.

Por ejemplo: En caso de hacer *click* cuando aparece el mensaje referido al cuerpo inferior del evaporador, se abre una ventana auxiliar para proporcionar al programa los datos sobre esta parte estructural. Esta acción queda representada en la Fig.2.

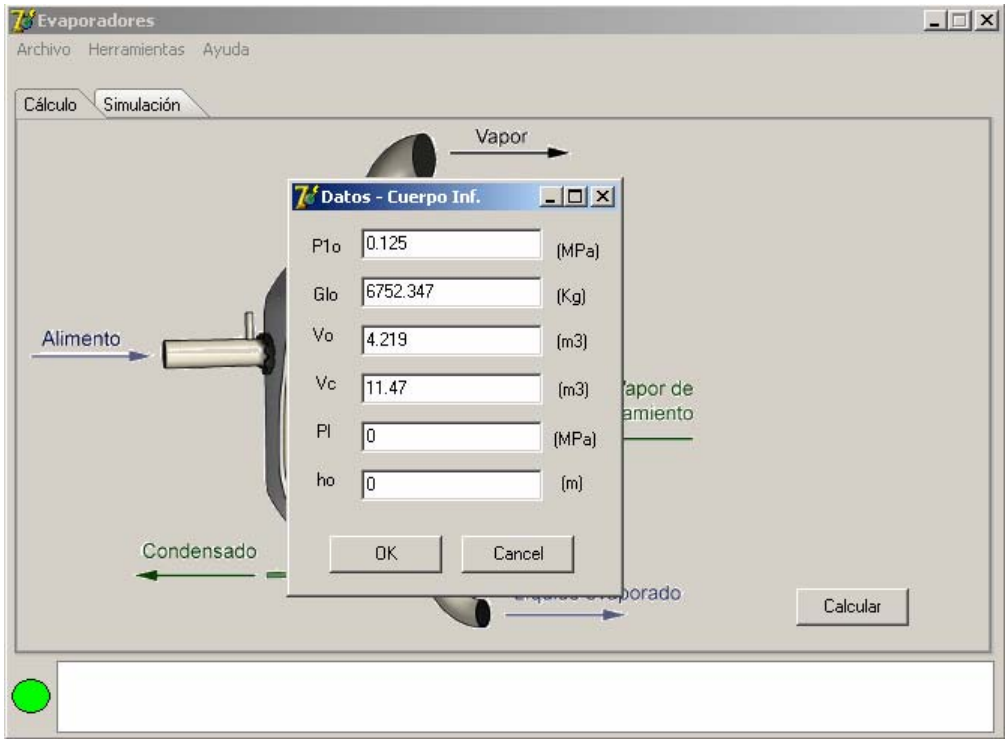

Fig2. Ventana para ingresar datos sobre el cuerpo inferior del evaporador.

Entrados todos los datos de manera similar a la descrita, el programa está en la capacidad de, oprimiendo el botón "Calcular", obtener todos los coeficientes de las matrices que describen el modelo en el espacio de estado del dispositivo estudiado, las cuales son fundamentales para la posterior simulación del mismo. Nuestro programa tiene la opción de guardar en disco los datos entrados, en forma de un fichero de texto. Esta operación se realiza accediendo al menú "Archivo" y escogiendo la opción "Guardar", con el objetivo de no tener que teclearlos nuevamente en una simulación posterior. Si por el contrario, lo que se desea es restituir cierto grupo de datos, previamente salvados, en su ubicación correspondiente dentro de cada ventana para la entrada de datos por teclado, se accede al menú "Archivo" y se escoge "Cargar". Estas operaciones hacen mucho más ágil la entrada de los parámetros y hacen los juegos de datos, más portables.

Una vez calculado el modelo se pasa a la otra hoja o "*TabSheet*" donde se observa el modelo calculado descrito de la forma:

$$\dot{x} = Ax + Bu + Fv$$

$$y = Cx$$

Para el caso del ejercicio tratado, la "Hoja de Simulación" muestra las ecuaciones matriciales del el modelo calculado según se muestra en la Fig.3.

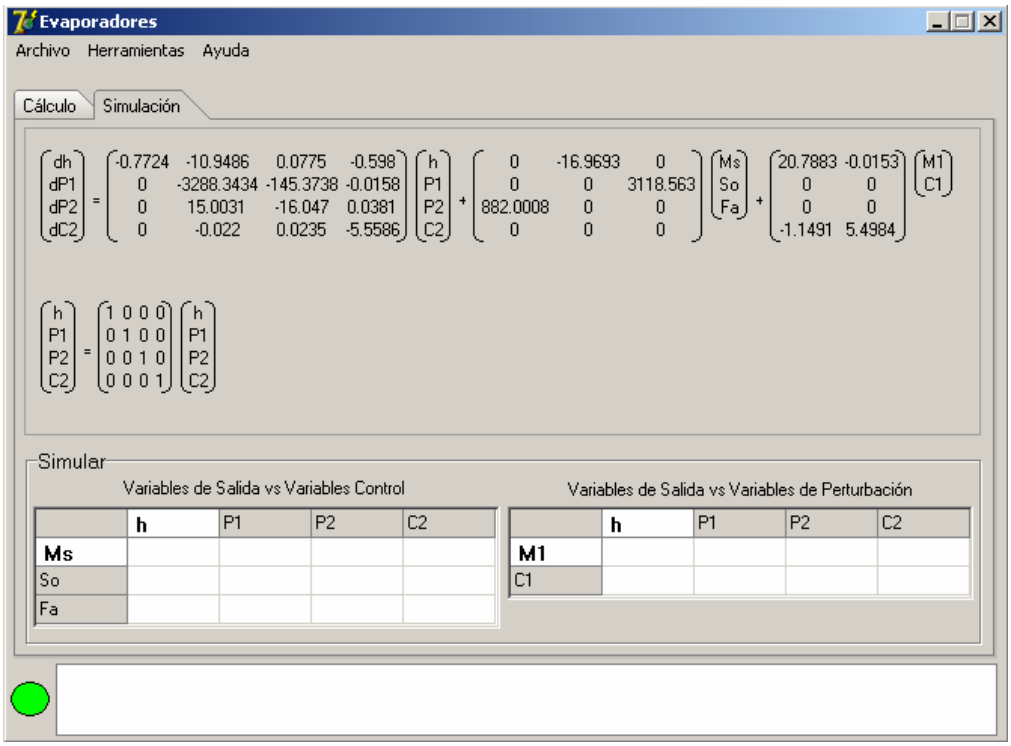

Fig.3: "Hoja de Simulación" del programa para el caso del ejercicio tratado.

Nótese que hasta el momento no ha habido necesidad de hacer ninguna consulta a tablas u otra fuente adicional de información sobre parámetros característicos de este proceso que, sin embargo, son necesarios para lograr la obtención de las matrices de estado. Listemos a continuación los **datos complementarios** necesarios para calcular el modelo de un evaporador de calandria vertical. Estos son:

- Densidad del flujo de jugo a la entrada y la salida en función de la concentración (den_1, den_2)
- Densidad del vapor de alimentación a la Calandria (den_s).
- Pendientes de las curvas que relacionan densidad vs. concentración del jugo a la entrada y la salida en los respectivos puntos de operación (d_1 y d_2)
- Pendiente de las curvas que relacionan densidad del vapor de alimentación vs. presión y temperatura en el punto de operación (d_3 y d_4)

- Pendiente de la curva que relaciona la densidad del vapor de alimentación a la presión vs. presión en la cámara superior (d_5)
- Calor latente para la evaporación a la presión en la cámara superior (λ)
- Calor latente para la evaporación a la presión en la cámara inferior (λ1)
- Variación de la temperatura más la elevación del punto de ebullición del jugo en función de la presión en la cámara superior ($\delta T_l / \delta P_1$)
- Variación de la temperatura de ebullición del jugo en función de la concentración de este ($\delta T_l / \delta C_2$)

Todos estos parámetros son función de los datos proporcionados en la orden del ejercicio, y se encuentran en tablas halladas en base a valores empíricos recogidos en la literatura [15] [8]. Algunos de estos datos no se averiguan a través de la simple inspección de dichas tablas, sino que es necesario hallar funciones y sus derivadas para ser evaluadas en los puntos de operación que correspondan. Este problema es resuelto al hacer uso de nuestra aplicación, pues los fragmentos que contienen los rangos que abarcan los puntos de operación normalmente utilizados en estos equipos, son cargados en MATLAB cada vez que éste es iniciado para el trabajo conjunto con nuestro programa. Un fichero llamado "TablasEvaporador.m" ubicado en una carpeta llamada "Tablas", la cual reside en el mismo subdirectorio donde está el ejecutable del programa, contiene dichos rangos declarados como vectores fila para, de forma automática, ajustar las funciones que describen su correspondencia, derivarlas, y evaluarlas en los puntos de operación que determinen los datos iniciales que se provean.

El alumno está ahora en condiciones de comenzar el proceso de simulación y el análisis de las respuestas obtenidas. Este, según se observa en la Fig.3, tiene la posibilidad de simular la respuesta de cualquier salida ante cualquier entrada, ya sean éstas pertenecientes al vector de mando o al vector de las perturbaciones. Supongamos que el alumno quiere simular la respuesta de P_1 ante una excitación tipo paso en M_s. Haciendo *click* en la casilla que relaciona P_1 con M_s se muestra el gráfico con la simulación correspondiente (Ver Fig.4).

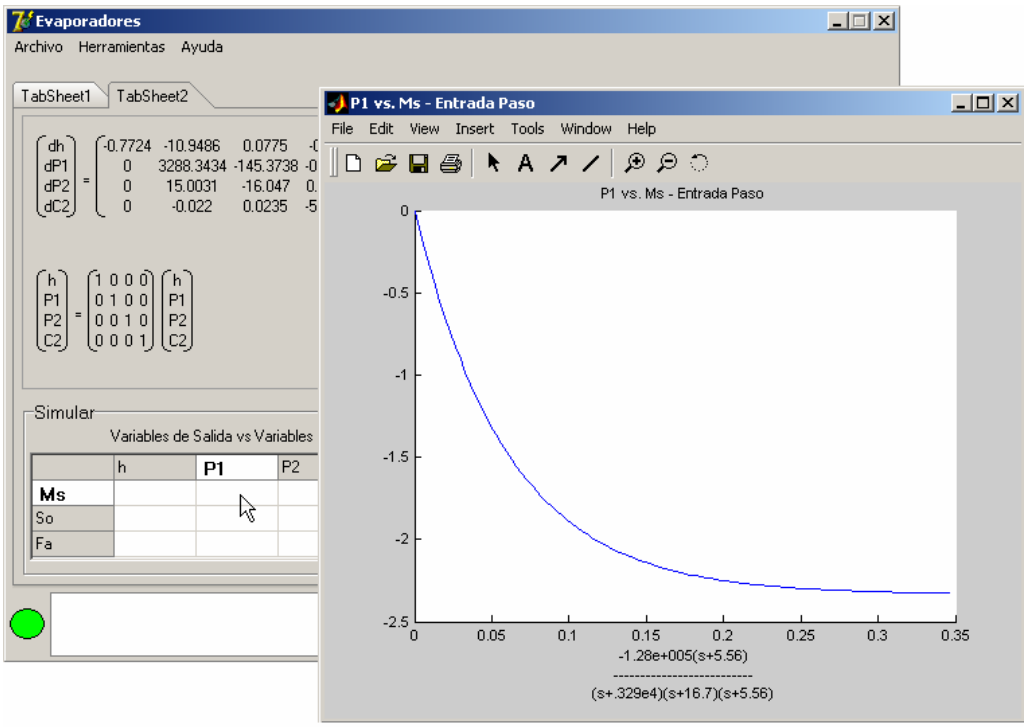

Fig.4: Resultados de la simulación de la respuesta de P_1 ante una entrada paso en M_s.

La representación gráfica de la respuesta transitoria está contenida dentro de una ventana con el aspecto similar a las que comúnmente fabrica MATLAB para ubicar espacios coordenados y plotear sobre ellos, pero con algunas modificaciones hechas para la mejor asimilación de la información que ofrecen. En el título de la ventana que contiene el gráfico, se muestra el par de variables a las que pertenece la simulación y en la parte inferior de la ventana se muestra la función de transferencia que relaciona al par entrada-salida simulado, descompuesta en ceros, polos y ganancia. Esto le permite al alumno, con toda la información a su disposición concentrada en un mismo gráfico, hacer un análisis más profundo sobre el modelo SISO que se tiene asumiendo líneas de entrada salida determinadas, validar este, y hacer reducciones si lo considera conveniente a partir de la identificación de los polos dominantes en la respuesta. En asignaturas posteriores pudiera utilizarlo también para conocer las interacciones entre los canales, determinar los mejores acoples y diseñar tanto los reguladores como los desacopladotes que sean necesarios.

Si el alumno ahora quiere simular la respuesta de *h* ante una entada paso en F_a, obtiene el resultado mostrado en la Fig.5.

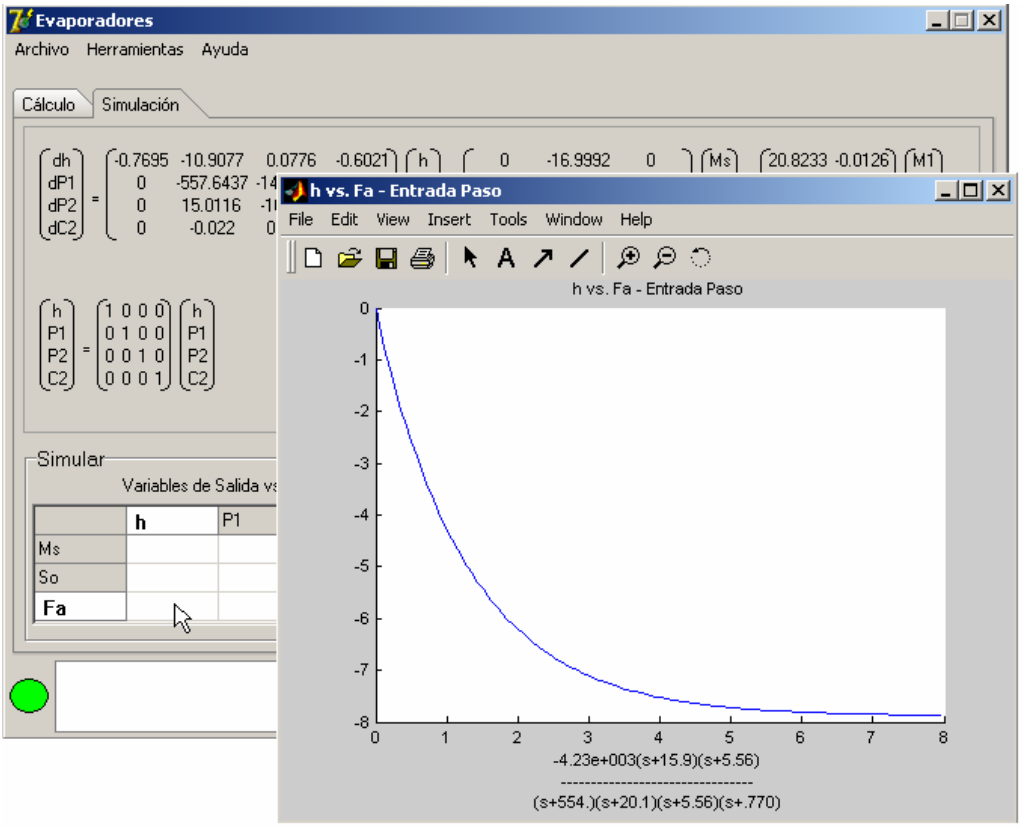

Fig.5 Resultados de la simulación de la respuesta de *h* ante una entrada paso en F_a.

Normalmente cada vez que se manda a simular un par entrada-salida cualquiera, el gráfico con la respuesta se aloja en la misma ventana, sobrescribiendo el anterior. En caso de que el alumno desee conservar cada gráfico en una ventana independiente, puede servirse de la opción "Múltiples Gráficos" del menú "Herramientas". Supongamos que se desee comparar la respuesta de C_2 ante cambios en C_1 y M_1 pero en gráficos separados, las ventajas del uso de esta opción se muestran en la Fig.6.

Por defecto, la ventana de comandos de MATLAB está oculta, no obstante se puede hacer visible mediante la opción "MATLAB Visible" del menú "Herramientas". Se recomienda extremar las precauciones a la hora de manipular alguna de las variables que ya están declaradas en el *workspace* pues el contendido de estas es vital para conservar la veracidad de los resultados que se obtengan en cada proceso de cálculo y simulación llevado a cabo por nuestra aplicación.

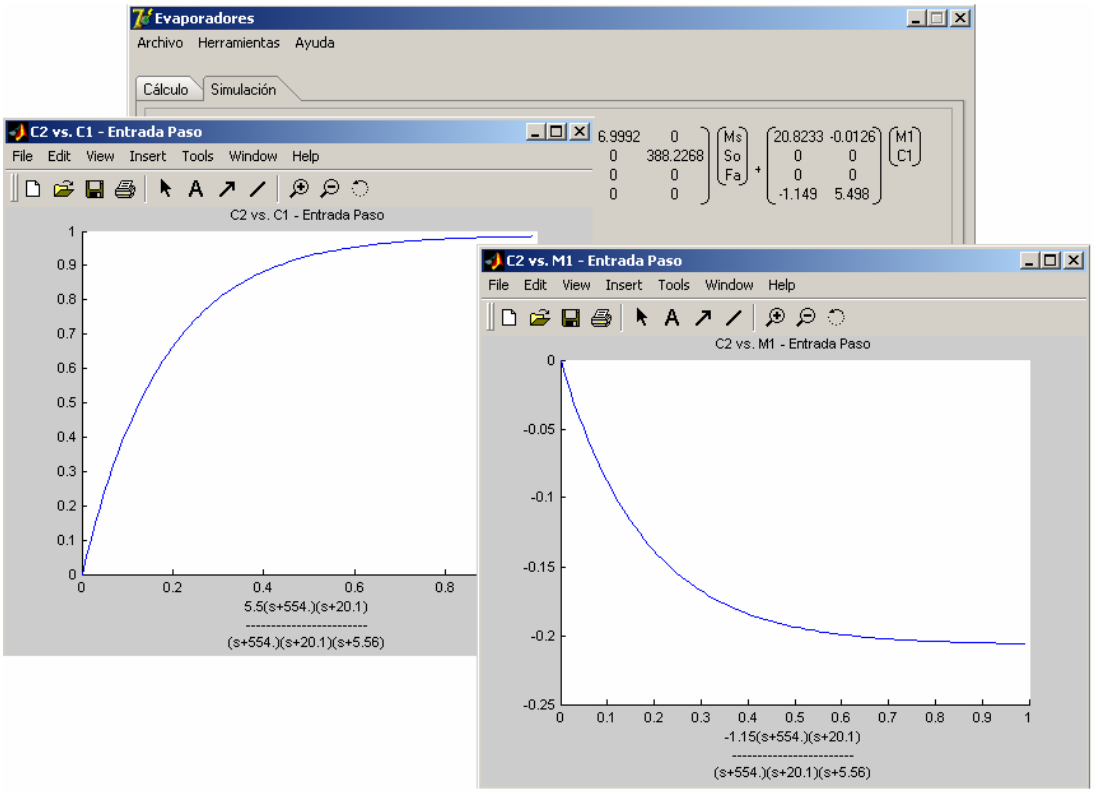

Fig.6: Ventajas de la opción "Múltiples Gráficos".

Si durante la ejecución de cualquier operación del programa, se detecta la no disponibilidad del servidor *Automation* de MATLAB, el cual se considera un error grave, se alertará al alumno de la situación y se le preguntara si desea relanzar MATLAB. En caso afirmativo se instancia un nuevo servidor *Automation* y con este una nueva sesión de MATLAB (Ver Fig.7).

3.3 Conclusiones

Se llega así al cumplimiento de las órdenes del ejercicio propuesto al inicio del capítulo. El alumno ha conseguido obtener los parámetros del modelo acorde con los datos proporcionados y simular todos los pares entrada-salida establecidos en él. No se requirió para esto la consulta de ninguna tabla de valores ni la realización de cálculos complementarios. El tiempo promedio para obtener el modelo y las respuestas que se deseen de este no debe exceder de 10 ó 15 minutos, considerando el peor caso en el que el alumno tenga que introducir los datos a través de teclado. Hasta este curso, cuando no se

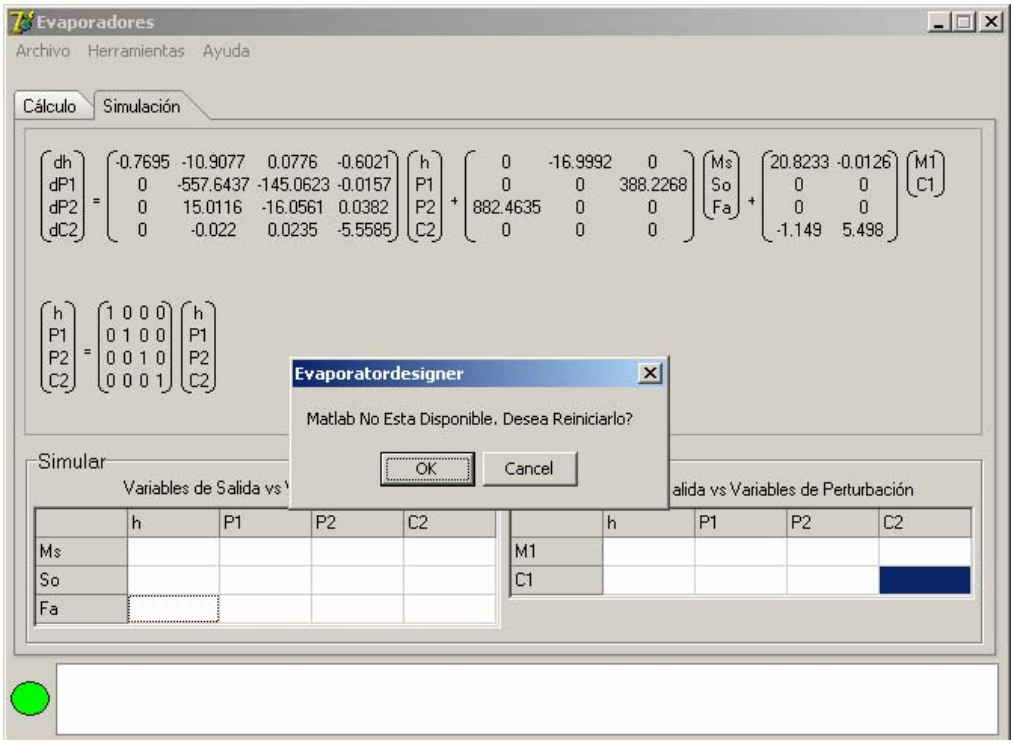

Fig.7: MATLAB no disponible.

días en alcanzar el mismo resultado. La posibilidad de inclusión de errores en el cálculo se reduce al mínimo, quedando restringida su ocurrencia solo a la duración de este proceso. La interfaz de la aplicación es muy intuitiva a la hora de entrar los datos, calcular y simular el modelo además de ofrecer un fácil acceso a todas las funcionalidades de esta.

Conclusiones

CONCLUSIONES

Al llegar al término de este trabajo podemos afirmar que:

- Ha quedado en manos de los estudiantes de nuestra especialidad, un software con todas las posibilidades para la fácil obtención del modelo de un evaporador de calandria vertical cuya complejidad como sistema multivariable, requiere de un considerable número de transformaciones matemáticas que entorpecen el cumplimiento del objetivo esencial de la propia modelación.
- Queda implementada la posibilidad de acceder a la simulación del comportamiento del sistema modelado en MATLAB, sin interactuar directamente con él.
- Para el caso que hemos descrito, se ha dotado a MATLAB de una interfaz más intuitiva y cómoda que evita el descenso en el ritmo de trabajo ante una tarea tan engorrosa.

Recomendaciones

RECOMENDACIONES

Demostrada la eficacia de una herramienta de este tipo en la enseñanza de las asignaturas de la disciplina de Sistemas de Control, y en la mejor comprensión de sistemas complejos, se sugiere la ampliación de las capacidades de uso que alcanzó la aplicación desarrollada con la culminación de este trabajo. Basados en el diseño de clases confeccionado con el propósito de acoger futuras expansiones del programa, sin modificar demasiado los módulos ya codificados, proponemos la inserción de otras estructuras en el diseño de clases que describan, según los requerimientos de la aplicación, otros dispositivos para ser calculados y simulados sus modelos.

Referencias Bibliográficas

REFERENCIAS BIBLIOGRAFICAS

[1] Aguado, Alberto. Enríquez, Jafet. Pascual, Jose M (1980). *Teoría del control moderno*. Editora de la Academia de Ciencias de Cuba, La Habana, Cuba.

[2] Anónimo (2001). *UML Applied – Object oriented Analysis and design using the UML,* Ariadne training limited. Disponible en www.ariadnetraining.co.uk consultado en Mayo 2004.

[3] Booch, Grady (1998). Classes and Objects. En: *Object - oriented anaysis and design with applications*. Second Edition. Rational. pp 81, Santa Clara, California.

[4] Borland Sofware Corporation (2002). *Delphi 7 User Manual*

[5] Chemical Engineering Department (2004). Basic Distillation Programs. Disponible en: http://unit-ps.che.ufl.edu/computer_tools/basic_programs/index.html, consultado: Junio 2004

[6] Cutiño Mendoza, Angel Y (2001). *Modelación y simulación del proceso de evaporación de cuádruple efecto en la industria azucarera*. Tesis de grado. Universidad Central "Marta Abreu" de las Villas, Santa Clara, Cuba.

[7] Henderson-Seller, B. y J.M. Edwards (1990). *The OO systems life cycle*. *Comm.Of ACM*, vol.33, no. 9.

[8] Meade, George P. Spencer, Emma F (1963). *Cane Sugar Handbook*, Ninth Edition, USA.

[9] Morris, Robert (1999), Multiple Effect Evaporator Online Calculations. Disponible en: http://www.sugartech.co.za/rapiddesign/multeffect/index.php3, consultado: Junio 2004

[10] Prosim Corp (2000).BatchReactor. Disponible en: http://www.prosim.net/kinetic.html, consultado: Junio 2004

[11] Prosim Corp (2000).BatchColumn. Disponible en: http://www.prosim.net/batch.html, consultado: Junio 2004

[12] Q. Kern, Donald (1969). *Procesos de transferencia de calor*. Edición Revolucionaria. La Habana, Cuba

[13] Rogerson, D. (1997). *Inside COM: Microsoft's Component Object Model*. Microsoft Press.

[14] Sparks Geoffrey, (2000). The Use Case Model.En*: An introduction to modelling software systems using the Unified Modelling Language*. Disponible en: www.sparxsystems.com.au, consultado: Mayo 2004

[15] Steam Tables. Vancouver, British Cplumbia, February 1978.

[16] The MathWorks, Inc.(2004), External Interfaces Reference. Disponible en: http://www.mathworks.com/access/helpdesk/help/techdoc/apiref

[17] The MathWorks, Inc.(2004), Advanced Users Guide.

[18] Zhyr, Joshua, *OLE Book*. Disponible en: \\172.20.2.1\Books\Bases de Datos\OLE, Red UCLV (Universidad Central "Marta Abreu" de las Villas), consultado: Abril 2004.