SISTEMA DE SUPERVISIÓN REMOTA DE SISTEMAS Y PROCESOS (SSRSP)

José Manuel Miguel Mínguez
Dpto de Informática y Automática, E.T.S.I. Informática, UNED, josemanuel.miguel@avanzit.com

Fernando Morilla García
Dpto de Informática y Automática, E.T.S.I. Informática, UNED, fmorilla@dia.uned.es

Resumen

El Sistema de Supervisión Remota de Sistemas y Procesos (SSRSP) es una aplicación cliente-servidor, orientada a facilitar la realización de pruebas (experiencias) a los alumnos de un laboratorio virtual y remoto de Automática y su seguimiento a los profesores. Permite el Análisis, Identificación y Monitorización de Procesos y Sistemas de Control a partir de medidas procedentes de ficheros o de servidores OPC. La aplicación está disponible para usuarios registrados, que sólo pueden acceder a la funcionalidad que tenga asignada su perfil (Administrador, Profesor o Alumno).

Palabras Clave: análisis de respuestas transitorias, identificación no paramétrica, laboratorio virtual y remoto, servidores OPC.

1 ANTECEDENTES

El Departamento de Informática y Automática de la UNED mantiene desde hace varios años una gran actividad investigadora ([1], [2] y [8]), encaminada a la puesta en marcha de un laboratorio virtual y remoto de Automática. Dos pilares fundamentales de este laboratorio son los entornos de experimentación basados en simulación y los entornos de experimentación remota ([7], [9] y [10]). Pero estos entornos no son autosuficientes, necesitan el complemento de herramientas para el análisis y diseño del sistema de control [6] y de otros recursos como los que se presentan en este trabajo para la estimación de modelos del proceso y para la supervisión y/o el análisis de las respuestas del sistemas de control.

Otro aspecto importante de un laboratorio virtual y remoto son las herramientas administrativas encaminadas a que el alumno sólo pueda realizar aquello para lo que está autorizado y que los profesores puedan consultar y corregir todo lo que sus alumnos hayan estado realizando. Este trabajo también aporta experiencias en esta dirección.

El núcleo del presente trabajo tiene su origen en la investigación, realizada por J. M. Miguel dentro del Programa de Doctorado "Automática e Informática industrial", donde se abordó la necesidad de sustituir el procesado "batch" de los datos medidos en el lazo de control, por un procesado "on-line" con toma de decisiones. Se eligieron entonces las dos situaciones siguientes, en las que se tenía una gran experiencia previa ([3] y [5]) y en las que el procesado "on-line" es de gran utilidad:

a. **Análisis de la respuesta transitoria de un sistema de control realimentado**. Este análisis es el paso fundamental en la estrategia de controlador autosintonizado representada en la Figura 1. Cuanto menos tiempo se emplee en determinar las características de la respuesta del sistema, más rápido se podrán calcular los nuevos parámetros de control y se podrá corregir cualquier situación no deseable que se haya producido en el lazo de control.

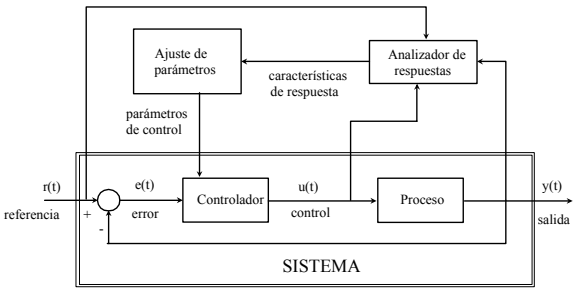

Figura 1: Controlador autosintonizado por análisis de la respuesta transitoria.

b. **Identificación de un modelo del proceso**. A diferencia del regulador autosintonizado de la Figura 1, la mayoría de los controladores autosintonizados hacen un uso explícito de un modelo paramétrico del proceso para calcular los parámetros de control. En ese caso, los parámetros del modelo del proceso se suelen identificar de forma recursiva a partir de los datos de entrada/salida, pero ésta no es la única forma. Algunos autores proponen explicitar la identificación de la respuesta escalón del proceso

(modelo no paramétrico), ya que su observación puede ayudar a la supervisión del proceso de identificación. La Figura 2 es un ejemplo de este tipo de identificación.

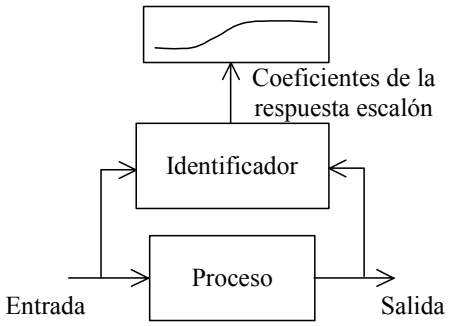

Figura 2: Identificación de un modelo del proceso

La investigación de J. M. Miguel se recogió en el documento [4], que describe un *Sistema* con esta doble funcionalidad. El *Sistema* está formado por dos instrumentos que reciben y procesan los datos provenientes de una máquina local. Cada instrumento consta de dos procesos: un emisor de datos (que obtiene las medidas leyéndolas de ficheros) y un receptor de datos (que procesa las medidas y hace el Análisis o la Identificación). Los dos procesos se comunican a través de un socket TCP/IP y pueden residir en la misma o en diferentes máquinas. Todos los procesos están implementados en lenguaje C++.

Al finalizar el trabajo de investigación se analizó la posibilidad de que el Sistema pasara a ser un instrumento del laboratorio virtual y remoto, para su uso vía Web, y comenzaron los desarrollos que se describen en el presente trabajo.

2 DESCRIPCIÓN DEL SISTEMA

2.1 PERFILES DE USUARIOS

La aplicación sólo está disponible para usuarios registrados, por tanto la entrada a la aplicación requiere una validación mediante 'usuario' y 'contraseña'. Pero además cada usuario disfruta únicamente de aquella funcionalidad que tenga asignada su perfil. Actualmente se han considerado tres perfiles posibles: *Administrador*, *Profesor* y *Alumno*.

2.1.1 Perfil Administrador.

El perfil *Administrador* es el único que puede realizar la gestión de usuarios y recursos del sistema. Mediante la opción de *Alta* puede crear un nuevo usuario, registrar sus datos personales y asignarle un nombre, una contraseña y un perfil. Si el perfil asignado es el de *Alumno*, podrá asignarle un profesor, limitar los módulos que puede ejecutar en las pruebas y los tipos de acceso a datos que puede tener.

Otras dos opciones (*Modificación* y *Consulta*) le permiten modificar o consultar los usuarios (todos o por perfiles) y sus datos, y la opción de *Baja* le permite eliminar el usuario y los datos registrados. La Figura 3 es un ejemplo de consulta realizada sobre uno de los cuatro usuarios con perfil de Alumno que existía en la base de datos, se observa que este usuario puede realizar pruebas con el IDENTIFICADOR pero no las puede realizar con el ANALIZADOR, y que los datos tienen que proceder de ficheros.

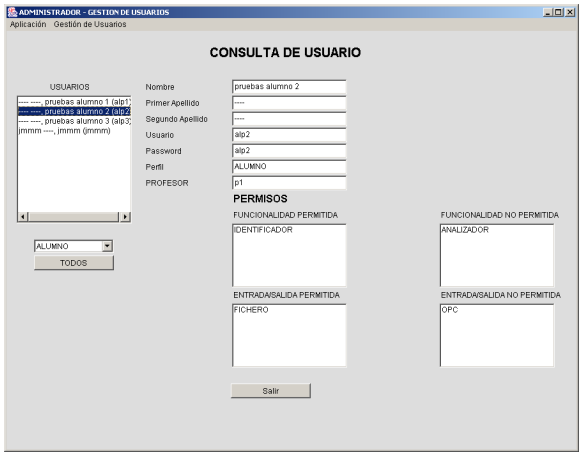

Figura 3: Ejemplo de consultas que puede realizar el *Administrador*

2.1.2 Perfil Profesor.

El usuario *Profesor* puede acceder a una relación de los alumnos que tiene asignados y a las pruebas que éstos hayan realizado. Seleccionando una de las pruebas realizadas por un alumno no sólo puede consultar bajo qué condiciones se realizó sino que además puede reproducirla.

El *Profesor* también dispone de toda la funcionalidad de pruebas, descrita en el apartado 2.2, pero a diferencia del *Alumno* sin ningún tipo de restricción.

2.1.3 Perfil Alumno.

El usuario *Alumno* accede a la funcionalidad descrita en el apartado 2.2, con las restricciones impuestas por el *Administrador*.

2.2 FUNCIONALIDAD

La aplicación se ha concebido con fines docentes y por tanto está orientada a facilitar la realización de pruebas (experiencias) a los alumnos y su seguimiento a los profesores. El alumno puede

realizar acciones sobre la prueba en curso o sobre las que ya tiene almacenadas en la base de datos. Tales acciones se describen en el apartado 2.2.1. Pero la funcionalidad más interesante de la aplicación reside en los módulos que reciben los datos y los procesan. Por el momento se han definido dos tipos de módulos (IDENTIFICADOR y ANALIZADOR), que se describen en los apartados 2.2.2 y 2.2.3. Y dos tipos de tecnologías para conectar con las fuentes de datos y con el receptor de resultados (FICHERO y OPC), que se describen en el apartado 2.2.4.

2.2.1 Gestión y ejecución de pruebas.

A través del interfaz de la Figura 4, el alumno puede: salir de la prueba en curso sin salvarla, guardar los cambios realizados en la prueba actual (estos cambios pueden afectar a los módulos que intervienen en la prueba o simplemente a la descripción de ésta), salvar la prueba en curso como una prueba nueva, borrar una prueba de la base de datos, abrir una prueba almacenada (recrea la prueba seleccionada, con los módulos y configuración que tiene asignados en la base de datos, y la deja preparada para ser ejecutada).

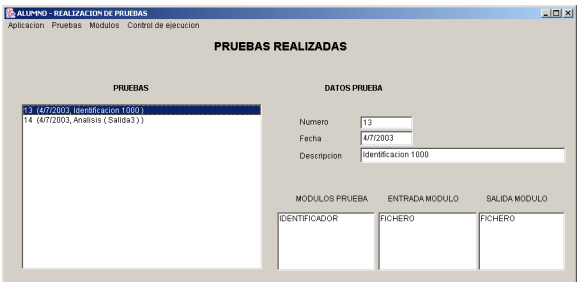

Figura 4: Interfaz del Perfil de Alumno

Pero lo más importante sin duda son los aspectos relacionados con la ejecución de la prueba; la aplicación incorpora opciones para: comenzar la prueba seleccionada, pararla (para el proceso de datos en cada módulo), continuarla (reanuda el proceso de datos), finalizarla (da por terminado el proceso de datos y por lo tanto la prueba) y resetearla (suspende la prueba y reinicia los interfaces gráficos).

La Figura 5 muestra un panel adicional que se ha incluido en la aplicación para que el usuario tenga agrupadas la mayoría de las acciones que puede realizar sobre la prueba en curso.

2.2.2 Módulo IDENTIFICADOR.

El módulo IDENTIFICADOR obtiene la respuesta impulsiva y la respuesta escalón de un sistema genérico (habitualmente un proceso) en base a un conjunto de medidas sucesivas de su entrada (habitualmente la señal de control). A este proceso se le denominó "*reconstrucción*" en el contexto del proyecto SINTOLAB.

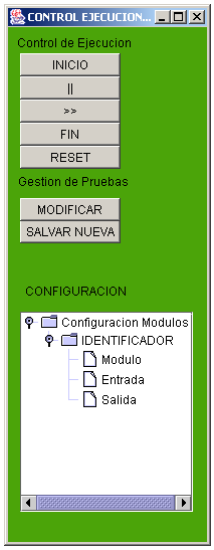

Figura 5: Panel que agrupa las principales opciones sobre la prueba en curso.

Tanto la respuesta impulsiva como la escalón respuestas están caracterizadas por un número M de coeficientes, que se calculan empleando el mismo método que en SINTOLAB [3]. Pero mientras que en SINTOLAB el proceso de identificación es totalmente *batch*, utilizando un número de medidas N+1 mucho mayor que el número de coeficientes, el módulo IDENTIFICADOR permite que el cálculo de coeficientes se realice así o continuamente desde el momento que se hayan recibido más de M+1 medidas. De esta forma, el usuario puede ver cómo evoluciona la identificación. el IDENTIFICADOR permite además que en el cálculo se utilicen únicamente las N últimas medidas de todas las recibidas hasta ese momento.

El cálculo de los coeficientes de la respuesta impulsiva G se basa en resolver el siguiente sistema de ecuaciones:

$$Y = U\,G\,\Delta\tau \quad \Rightarrow \quad G = \frac{1}{\Delta\tau} U^{-1}\,Y$$

donde

Y es un vector columna (matriz Nx1) que se forma con las medidas incrementales de la salida del sistema

$$Y^T = [y(1)-y(0),\ y(2)-y(1),\ \ldots,\ y(N)-y(N-1)]^T$$

U es una matriz NxM formada con las medidas de la entrada al sistema

$$U = \begin{bmatrix} u(1)-u(0) & u(0) & \dots & 0 \\ u(2)-u(1) & u(1)-u(0) & \dots & 0 \\ . & . & \dots & . \\ u(M-1)-u((M-2) & u(M-2)-u(M-3) & \dots & u(0) \\ u(M)-u(M-1) & u(M-1)-u(M-2) & \dots & u(1) \\ . & . & \dots & . \\ u(N)-u(N-1) & u(N-1)-u(N-2) & \dots & u(N-M+1) \end{bmatrix}$$

G es un vector columna (matriz Mx1) que contiene los valores (coeficientes) de la respuesta impulsiva del sistema

$$G^T = [g(1), g(2), \dots, g(M)]^T$$

$\Delta\tau$ es el intervalo de muestreo

U^{-1} representa la matriz pseudoinversa de la matriz U, obtenida mediante descomposición en valores singulares tal como se describe en las *Numerical Recipes* [11]

En la Figura 6 se muestra la interfaz gráfica del IDENTIFICADOR, a través de la cual el usuario puede parametrizar la identificación y va a recibir información gráfica de cómo evolucionan las señales del sistema y las estimaciones (respuesta impulsiva y respuesta escalón). Los resultados de la identificación dependen de los siguientes parámetros: la *ventana* de medidas (número máximo de medidas que se procesan), el *tiempo de muestreo*, el *número de coeficientes* a calcular y el *método de cálculo*.

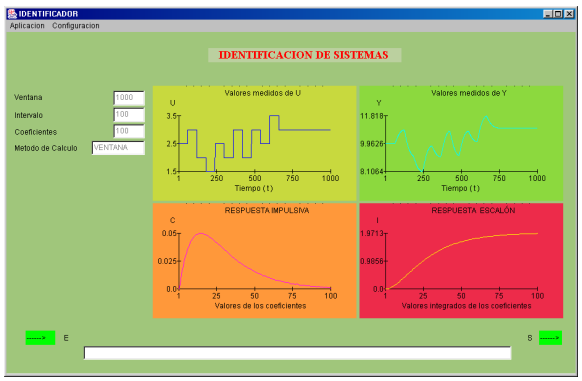

Figura 6: Interfaz del módulo IDENTIFICADOR.

Se consideran tres posibles métodos de cálculo:

VENTANA; El cálculo de coeficientes se realiza una sola vez, cuando se han recibido las medidas indicadas en el campo *ventana*.

DINÁMICO; El cálculo de coeficientes se realiza con todas las medidas recibidas hasta un instante de muestreo. Comienza cuando se ha recibido un número de medidas igual al *número de coeficientes+1* y finaliza cuando se han recibido las medidas indicadas en el campo *ventana*.

CONTINUO; Es como el método *DINÁMICO*, pero el cálculo de coeficientes no finaliza cuando se han recibido las medidas indicadas en el campo *ventana* sino que continúa indefinidamente utilizando las últimas medidas, tantas como indica la *ventana*.

2.2.3 Módulo ANALIZADOR.

El módulo ANALIZADOR procesa las medidas sucesivas de la señal de error de un sistema de control realimentado, determina la naturaleza de la respuesta y los parámetros que la caracterizan:

- Tipo de respuesta; no oscilatoria por cambio en la consigna, oscilatoria por cambio en la consigna, no oscilatoria por cambio en la carga, oscilatoria por cambio en la consigna.
- Si se ha alcanzado o no el estado estacionario.
- Tiempo de asentamiento.
- Si existe o no existe sobreelongación.
- La máxima sobreelongación.
- El tiempo de pico.
- Si se ha podido medir o no la razón de amortiguamiento.
- La razón de amortiguamiento.
- El pseudoperiodo de oscilación.

Pero a diferencia del proceso de análisis tipo *batch* utilizado en SINTOLAB [3] y en [5], donde se pueden presentar incluso más características de las reseñadas anteriormente, el ANALIZADOR es un autómata de estados, similar al empleado por el regulador EXACT, capaz de presentar además resultados intermedios en dos momentos concretos del análisis:

Salida *S1*; Cuando ha detectado los tres picos, véase como ejemplo la Figura 7, y ha determinado el tipo de respuesta, el tiempo de pico, la razón de amortiguamiento, el pseudoperiodo de oscilación y los valores de los tres picos.

Salida *S2*; Cuando ha detectado el estado estacionario, y como consecuencia de ello se conoce el tamaño de la banda utilizada para validar el estado estacionario, el tiempo de asentamiento y la máxima sobreelongación.

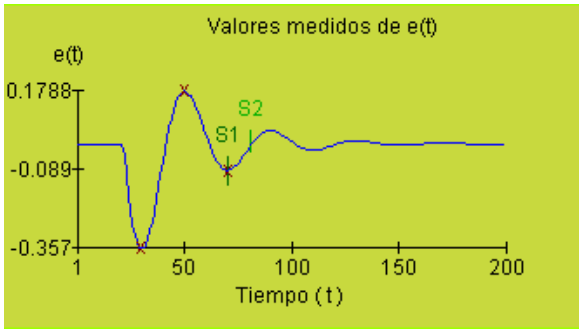

Figura 7: Ejemplo de análisis mediante búsqueda de los tres picos.

En la Figura 8 se muestra la interfaz gráfica del ANALIZADOR, a través de la cual el usuario puede parametrizar el análisis y va a recibir información gráfica de cómo evoluciona la señal de error del sistema de control realimentado, cómo evoluciona el análisis (esquematizado por el correspondiente autómata de estado y por las características determinadas). Los resultados del análisis dependen de los siguientes parámetros: la *ventana* de medidas (número máximo de medidas que se procesan), el *tiempo de muestreo*, el *margen* de error empleado para validar el estado estacionario y el *método de cálculo*. El usuario dispone de dos métodos de cálculo:

VENTANA; El análisis se realiza una sola vez, cuando se han recibido las medidas indicadas en el campo *ventana*.

CONTINUO; El análisis se realiza por primera vez cuando se han recibido las medidas indicadas en el campo *ventana* y se repetirá indefinidamente con las medidas que sigan llegando hasta completar sucesivas ventanas.

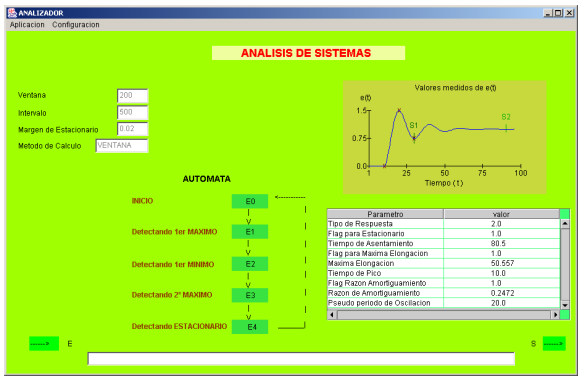

Figura 8: Interfaz del módulo ANALIZADOR.

2.2.4 Fuentes de datos y receptor de resultados.

Los módulos funcionales reciben los datos procedentes de unos emisores (fuentes de datos), los procesan, presentan resultados durante el procesamiento y los envían al receptor correspondiente. El usuario está obligado a configurar la ENTRADA y la SALIDA de los módulos para indicar cuál va a ser la procedencia de los datos y dónde quiere registrar sus resultados. En la actualidad están implementadas dos tecnologías de ENTRADA/SALIDA: *FICHERO*, que permite obtener los datos de ficheros y salvar los resultados en un fichero, y *OPC*, que permite obtener los datos y enviar los resultados a través de un servidor OPC.

Si la ENTRADA o la SALIDA se eligen de tipo *FICHERO*, basta con seleccionar los ficheros del PC local que contienen las medidas o el fichero que va a recibir los resultados. La Figura 9 es un ejemplo de ENTRADA tipo *FICHERO*, donde se observa que el usuario ha tenido que especificar dos fuentes de datos, el fichero de la señal de control y el de la salida del sistema, existentes en el disco duro de su PC.

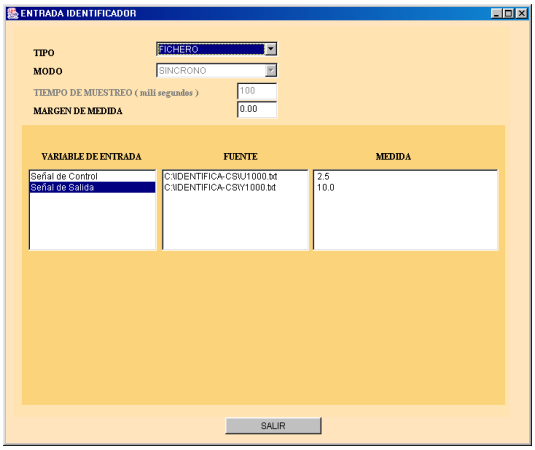

Figura 9: Configuración de *ENTRADA* tipo *FICHERO* para el IDENTIFICADOR.

Si la ENTRADA o la SALIDA se eligen de tipo *OPC*, hay que seleccionar el Host donde está el servidor OPC, el servidor concreto y los ítems que sirven las variables medidas o los ítems en los que se reciben los resultados. La ausencia de Host se interpreta como que el servidor OPC reside en el PC del usuario. Si se configura la *ENTRADA*, véase como ejemplo la Figura 10, también se debe seleccionar el margen de las medidas dentro del cual dos medidas se consideran iguales y el método de obtención: SÍNCRONO ó ASÍNCRONO. En el método ASÍNCRONO el servidor OPC envía a la tarea de *ENTRADA* (que actúa como cliente OPC) la actualización de las medidas sólo cuando cambian. En el método SÍNCRONO será la *ENTRADA* la que solicite las medidas al servidor OPC cada vez que las necesite la tarea de *PROCESO*.

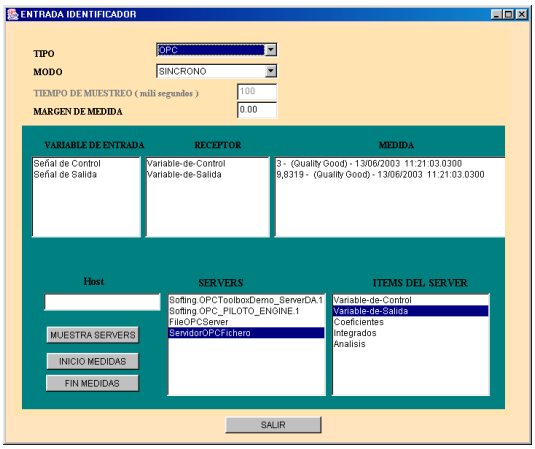

Figura 10: Configuración de *ENTRADA* tipo *OPC* para el IDENTIFICADOR.

3 DISEÑO DEL SISTEMA

La aplicación reside íntegramente en un servidor WWW del laboratorio, por lo que los usuarios pueden utilizarla remotamente, desde su computador personal a través de Internet, o localmente, desde los puestos de trabajo del laboratorio. En el cliente debe estar instalado un navegador web (Netscape 7.0, Explorer 6.0 o superiores) con máquina virtual JAVA para la ejecución de applets y se deben haber descargado las librerías que usa la aplicación: *Coeficientes.dll* para el cálculo de coeficientes, *DLLClienteOPCJava.dll* y *WTclient.dll* para el uso de conexiones OPC. En el servidor, véase Figura 11, se encuentran todos los ficheros y componentes de la aplicación: paginas web de acceso e información, clases Java que implementan la funcionalidad deseada, clases de acceso a la Base de Datos y la Base de Datos del Sistema.

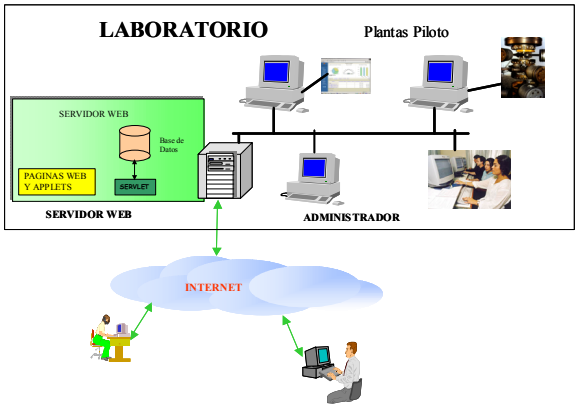

Figura 11: Localización de la aplicación

El usuario se debe validar mediante un nombre de usuario y una contraseña. Una vez autorizado el acceso, se descargan automáticamente las clases JAVA necesarias para cubrir la funcionalidad correspondiente al perfil que tiene asignado.

El acceso a Base de Datos se realiza mediante un modelo cliente-servidor. El cliente es un *applet* de la aplicación (descargado por el usuario) y el servidor es un *servlet*, que reside en el servidor Web, y que gestiona todos los accesos a la Base de Datos. La comunicación entre el *applet* y el *servlet* se realiza mediante conexiones HTTP establecidas por iniciativa del applet.

El acceso a las Plantas Pilotos del laboratorio, se realizará con un modelo similar al anterior, en el que existirá un *servlet* que gestionará el acceso a estas Plantas mediante una pasarela OPC.

3.1 ARQUITECTURA DE LA APLICACIÓN

La aplicación consta de los cuatro elementos (Acceso, Administrador, Profesor y Alumno) representados en la Figura 12, cada uno de los cuales realiza una funcionalidad concreta.

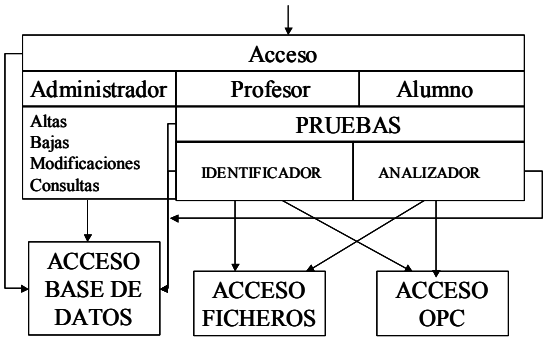

Figura 12: Arquitectura de la aplicación.

El elemento *Acceso* se encarga de presentar un interfaz a través del cual el usuario introduce su nombre de usuario y contraseña, los recoge y hace la validación accediendo a la Base de Datos. Si la validación es correcta, obtiene el perfil del usuario y lanza el interfaz correspondiente a dicho perfil.

El elemento *Administrador* implementa el interfaz correspondiente a este perfil y toda la funcionalidad, descrita en el apartado 2.1.1, que hace referencia a las Altas, Modificaciones, Consultas y Bajas de usuarios.

El elemento *Profesor* implementa el interfaz correspondiente a este perfil y toda la funcionalidad, descrita en el apartado 2.1.2, que hace referencia a la consulta de pruebas de alumnos y la posibilidad de realización de pruebas propias.

El elemento *Alumno* implementa el interfaz correspondiente a este perfil y toda la funcionalidad, descrita en el apartado 2.2.1, que hace referencia a la gestión, preparación y realización de las pruebas.

3.2 ESTRUCTURA DE LOS MÓDULOS

Aunque cada módulo funcional de la aplicación tiene sus propias características y su propia interfaz gráfica, todos responden a la misma estructura interna de la Figura 13, con tres partes fundamentales e independientes entre sí: el *PROCESO* que es el que realmente caracteriza al módulo y su funcionalidad (para el IDENTIFICADOR es la descrita en el apartado 2.2.2, y para el ANALIZADOR la descrita en el apartado 2.2.3), la *ENTRADA* que se encarga de la adquisición de datos o medidas y la *SALIDA* que es la que gestiona el envío de resultados. De esta forma se aísla la recepción y el envío de datos de su

procesamiento, se facilita la posibilidad de ampliación de la herramienta con nuevos módulos y con nuevos tipos para acceder a las medidas o entrega de datos. Para incluir un módulo con una nueva funcionalidad bastaría con crear su interfaz gráfica y su parte de *PROCESO*.

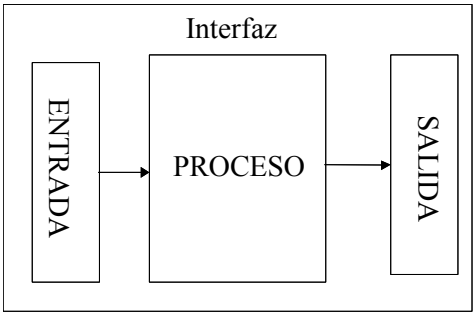

Figura 13: Estructura de un módulo funcional

La relación entre las tres partes se realiza mediante un interfaz interno que siempre se invoca por iniciativa de la tarea de *PROCESO*. Éste indica a la *ENTRADA* el nombre y número de variables que necesita recibir y a la *SALIDA* el número y nombre de variables que va a suministrar. Para recibir medias básicamente invoca las siguientes primitivas: *InicioMedias* (para que la *ENTRADA* se conecte con el suministrador de medidas), *DameMedidas* (la *ENTRADA* debe obtener las medidas y proporcionarlas en un vector), *FinMedidas* (la *ENTRADA* debe dejar de recibir medidas o desconectarse de la fuente). Para enviar resultados se invoca la primitiva *PonDatos*, que indica a la *SALIDA* que envíe los datos al receptor.

3.3 CÁLCULO DE COEFICIENTES EN EL MÓDULO IDENTIFICADOR

Como se indicó en el apartado 2.2.2, el cálculo de los coeficientes se realiza mediante el Método Matemático basado en la Descomposición en Valores Singulares propuesto en el Numerical Recipes. Ante la dificultad de migrar el algoritmo de cálculo (que estaba implementado en C++, [4]) a lenguaje JAVA, se ha optado por encapsular la funcionalidad de cálculo de coeficientes en una DLL que puede ser cargada desde una clase JAVA. Para hacer esto se ha empleado el interfaz JNI, se ha creado una clase que define el interfaz de la librería, se ha exportado este interfaz a un fichero cabecera de C++ (con las herramientas de JNI) y con esta cabedera se creado la librería DLL compilada en C++. El resultado es óptimo y el cálculo se realiza correctamente.

3.4 DISEÑO DEL ACCESO OPC

El software que implementa el estándar OPC está realizado en gran medida en lenguaje C++ y esto hace que en general se pueda únicamente tener acceso a librerías de este entorno. Para implementar el acceso a servidores OPC desde JAVA se ha realizado un diseño similar al descrito en el apartado 2.3.3.

Como software base de acceso a OPC, implementado en C++, se ha elegido una librería de la empresa Wintech (http://www.win-tech.com/). Esta librería proporciona la gestión de conexiones con los servidores y el intercambio de datos bajo estándar OPC. Nuestro trabajo ha consistido en explotar esa capacidad y recubrirla para que pueda ser usada por uno o varios clientes JAVA simultáneamente. Se ha creado un interfaz sencillo para acceso a los servidores OPC y los ítems. También se realiza el mapeo de datos al formato intercambiado en OPC (*VARIANT*).

La estructura para el acceso de clientes JAVA es la indicada en al Figura 14.

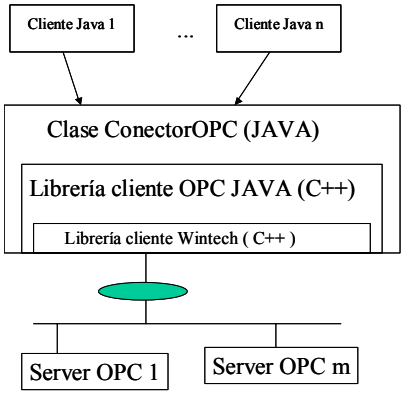

Figura 14: Estructura del acceso OPC.

Esta estructura permite por una parte realizar la integración JAVA/C++ y por otra que cada nivel proporcione al nivel superior una mayor abstracción de los datos intercambiados. El interfaz final que ofrece la clase JAVA *ConectorOPC* a un cliente se compone básicamente de las siguientes primitivas:

- boolean java_obtieneServers(String host, Vector servidores, String outError).
- boolean java_obtieneItemsServer(Vector items, String outError)
- boolean java_ponItemServer(String item, String outError)
- boolean java_inicioMedidasServer(String tipoRecepcion, long tiempoMuestreo, float margenMedidas, String outError)
- boolean java_finMedidasServer(String outError)
- boolean java_escribeMedidaItemServer(Vector valor, int tipo, String outError)

El mapeo de los valores manejados en la aplicación a los intercambiados en OPC (tipo *VARIANT*) se realiza en la *Librería cliente OPC Java*. En los datos simples, como las medidas, el valor recibido es volcado a un *String* y se pasa a la clase *ConectorOPC* con este tipo. En la escritura de los valores obtenidos, la librería los recibe como un *Array* de valores del tipo *double* y para ser escritos en el servidor OPC los convierte a un *Array* del tipo *SAFEARRAY*:

```
VARIANT *pVal
SAFEARRAY *pSA;
SAFEARRAYBOUND aDim[1];
        .....
 pSA= SafeArrayCreate(VT_VARIANT,1,aDim);
        .....
pVal->vt= VT_ARRAY | VT_VARIANT;
pVal->parray= pSA;
```

Servidor OPC de pruebas

Para probar la funcionalidad descrita anteriormente se ha desarrollado también un servidor OPC (Figura 15) que es capaz de suministrar medidas de dos variables a través de dos ítems, cuyos valores se leen de sendos archivos. Al mismo tiempo que ofrece tres ítems de escritura de tipo Array.

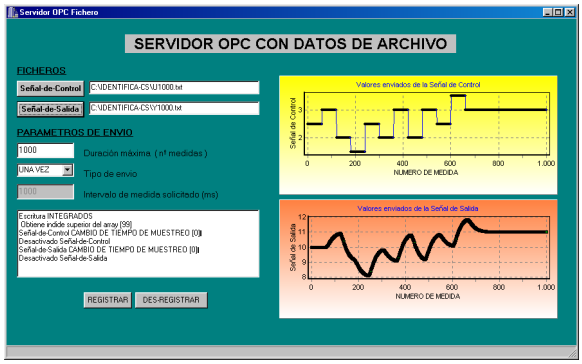

Figura 15: Servidor OPC de pruebas.

3.5 DISEÑO DEL ACCESO A LA BASE DE DATOS

Como se comentó al comienzo de la sección 3, el acceso a la Base de Datos se realiza mediante un modelo cliente-servidor donde el cliente es un *applet* de la aplicación y el servidor es un *servlet* que reside en el servidor Web.

La comunicación entre el *applet* se realiza a través una conexión sobre el protocolo HTTP, siguiendo los pasos que se describen a continuación. Se crea la *URL* para la conexión

```
URL pagina = miApplet.getCodeBase();
String protocolo = pagina.getProtocol();
String servidor = pagina.getHost();
int puerto = pagina.getPort();
String servlet = "/ssrsp/servlet/mi_servlet";

  URL direccion = new URL ( ..... )
```

Después se crea una *URLConnection* con la dirección dada y se modifican las propiedades: se indica que no use caché, que es conexión de entrada y salida, etc.. Se crea un flujo de datos del tipo *ObjectOutputStream* para enviar los datos al servidor y se envían en un *Vector*.

Para recibir la respuesta del servidor se abre un flujo de entrada del tipo *ObjectInputStream* y se reciben los datos en un *Vecto*r.

Se han definido una serie de primitivas para el intercambio de información. El cliente envía al servidor un vector donde el primer elemento es la primitiva y los siguientes datos necesarios asociados a la primitiva. El servidor recibe la petición, analiza la primitiva y realiza el acceso a la Base de Datos. La respuesta a la petición es un vector que envía al cliente, indicando en el primer elemento si la operación se ha realizado de forma correcta o no, y si es correcta envía el resultado.

Por ejemplo en el acceso para obtener los datos de configuración almacenados para la ENTRADA de un Módulo IDENTIFICADOR, el applet cliente envía un vector con la primitiva *"LEE_DATOS_CONF"* en la primera fila, en la segunda fila irá el número de prueba, en la tercera el tipo de módulo (*"IDENTIFICADOR"*) y en la cuarta la parte del módulo (que en este caso concreto será *"ENTRADA"*). El servlet servidor identificará la primitiva y formará la sentencia SQL correspondiente. Si la *ENTRADA* es de tipo OPC, el resultado de la consulta será un vector con la primera fila al valor "OPC", en la segunda el modo de obtención de medidas ("SÍNCRONO" o "ASÍNCRONO"), en la tercera el margen de medias (p.e. 0.0), en la cuarta el *Host* donde está el servidor OPC, en la quinta el nombre del servidor OPC, en la sexta el ítem del cual se lee la señal de control y en la séptima el ítem del cual se lee la señal de salida.

JDBC para acceso a la Base de Datos

Para el acceso a la Base de Datos se emplea el interfaz JDBC, que permite a los programas Java conectar con una gran variedad de bases de datos. Esto permite mantener una única programación de los accesos a la base de datos independientemente de la plataforma donde resida el sistema y de la base de datos empleada. Únicamente se tendrán que configurar los parámetros de la conexión: la URL

donde se encuentra la base de datos, el usuario, la clave y el driver empleado.

En nuestro caso, como la plataforma del laboratorio tiene Sistema Operativo LINUX y base de datos POSTGRE, los parámetros están configurados de la siguiente manera :

URL = "jdbc:postgresql://localhost:5432/ssrspbd"
DRIVER = "org.postgresql.Driver"

En la maqueta de pruebas que tiene Sistema Operativo Windows 98 y base de datos ACCES, la configuración es:

URL = "jdbc:odbc:ssrspbd"
DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver"

4 CONCLUSIONES

En el momento actual se dispone de una aplicación a la que se accede vía Web, en el que se han implementado los tres perfiles de acceso (Administrador, Profesor y Alumno) y se han completado las funcionalidades de los módulos IDENTIFICADOR y ANALIZADOR.

Se ha implementado la toma de datos desde ficheros y el salvado de los resultados a ficheros. Esto permite que el alumno pueda hacer pruebas con datos propios o datos que le son suministrados por el profesor y obtener resultados para diferentes parametrizaciones de los módulos.

Se ha implementado el acceso a datos y resultados mediante el estándar OPC. Esto permite que varios módulos o clientes puedan estar simultáneamente accediendo tanto a las medidas suministradas como a los resultados obtenidos. Se dispone de un servidor OPC de pruebas que suministra los datos que obtiene de ficheros y recoge los resultados en vectores. Para el usuario serán visibles todos los servidores OPC que se encuentren en la máquina desde donde ha conectado a la aplicación, o en las máquinas de la misma red local (siempre que los servidores estén arrancados y se hayan configurado adecuadamente los permisos de acceso).

4.1 FUTURAS AMPLIACIONES

En breve se pretende incluir un módulo MONITOR que sea capaz de visualizar gráficamente las variables que se deseen y sean suministradas por un servidor OPC. Se pretende incluir un nuevo *servlet* en el servidor, con la funcionalidad necesaria para que el alumno pueda acceder a una batería de ficheros de datos, suministrados por el profesor y que residirán en el propio servidor Web (en vez de en la máquina local).

Se pretende también dotar a las plantas del laboratorio de un servidor OPC. Para acceder a estas plantas se creará un nuevo tipo de conexión OPC *REMOTA*, que por una parte se conectará con la base de datos para saber qué plantas están accesibles y a continuación con un proceso que hará las funciones de pasarela. De esta forma se evita el acceso directo a las plantas y se pueden gestionar, a través de la base de datos, las plantas e ítems a los que cada usuario tendrá acceso.

Se prevé poder incluir nuevos tipos de acceso a datos (ActiveX, conexión TCP/IP, etc..), que faciliten la conexión con otras aplicaciones, con los que se pueda interactuar y ampliar la funcionalidad.

Agradecimientos

A nuestro compañero Rafael Pastor que nos ha ayudado a montar la plataforma del laboratorio y a poner a punto la base de datos. A Jesús M. Zamarreño de la Universidad de Valladolid por su asesoramiento en la búsqueda de información sobre OPC. Y a los compañeros de J. M. Miguel en AVANZIT TECNOLOGÍA por la ayuda prestada siempre que la ha necesitado.

Este trabajo ha sido financiado por la CICYT a través del proyecto DPI 2001-1012.

Referencias

[1] Dormido, S., J. Sánchez, F. Morilla (2000). "Laboratorios virtuales y remotos para la práctica a distancia de la Automática", Sesión Plenaria, Actas de las *XXI Jornadas de Automática*, Sevilla 18, 19 y 20 de septiembre.

[2] Dormido, S. (2002). "Control Learning: Present and Future", *b´02 15th World Congress of IFAC*, Barcelona, 21-26 july 2002.

[3] González-Martín, R., F. Morilla, I. López, R. Pastor (2003). "Sintolab: The Repsol-YPF PID tuning tool", *Control Engineering Practice*, May.

[4] Miguel, J. M. (2001). "Supervisión remota de procesos o sistemas de control". Trabajo de investigación dentro del Programa de Doctorado "Automática e Informática Industrial" del Departamento de Informática y Automática de la UNED.

[5] Morilla, F., N. Duro, A. González (2000). "Auto-tuning PID Controllers in terms of

relative camping", *Past, present and future of PID Control*, J. Quevedo and T. Escobet (Eds.), pp. 161-166.

[6] Morilla, F., A. W. Fernández, S. Dormido Canto (2001). "Control systems análisis & design server", *Workshop on Internet Based Control Education IBCE'01*, Madrid, December 12 –14.

[7] Morilla, F., A. Isabel, J. Sánchez (2002). "Entorno de experimentación sobre control de nivel y control de caudal", *XXIII Jornadas de Automática*, La Laguna 9-11 de septiembre.

[8] Sánchez, J. (2001). Un nuevo enfoque metodológico para la enseñanza a distancia de asignaturas experimentales: análisis, diseño y desarrollo de un laboratorio virtual y remoto para el estudio de la Automática a través de Internet. Tesis doctoral, UNED.

[9] Sánchez, J., F. Morilla, S. Dormido (2001). "Teleoperation of an inverted pendulum through the world wide web", *Workshop on Internet Based Control Education IBCE'01*, Madrid, December 12 –14.

[10] Sánchez, J., F. Morilla, S. Dormido, J. Aranda and P. Ruipérez (2002). Virtual Control Lab using Java & Matlab: A qualitative aproach. IEEE Control System Magazine, vol. 22, no. 2, pp. 8-20.

[11] Willian H. Press, Willian T. Vetterling, Saul A. Teukolsky, Brian P. Flannery. *Numerical Recipes : Example Book (C), Second Edition y Numerical Recipes in C (The Art of Scientific Computing) Second Edition.*