Una Arquitectura Integrada para la implementación de Laboratorios Remotos de Control Automático

Zavalla, Eduardo; Fernandez, Arnoldo; Orellana, Adrián
(ezavalla, orellana)@inaut.unsj.edu.ar; arnoldo@unsj.edu.ar
Instituto de Automática – Facultad de Ingeniería
Universidad Nacional de San Juan

Resumen

Desde hace mas de una década se ha podido apreciar una marcada tendencia a la utilización de laboratorios remotos como medio de brindar aplicación práctica de los contenidos para los alumnos que participan de las metodologías de Educación a Distancia. Sin embargo, aunque hay muchas variantes de los sistemas de información utilizados para esta tarea, todos ellos confluyen en un conjunto de dificultades relacionadas con la administración de los LMS utilizados y con la complejidad derivada de la no-utilización de las mismas herramientas usadas en la enseñanza para la materialización de los recursos prácticos. Este trabajo propone una arquitectura para la implementación de laboratorios remotos de control automático integrada a las habilidades docentes, de forma tal de reutilizar y potenciar la experiencia teórica de los docentes en la creación de los componentes prácticos que puedan ser puestos fácilmente a disposición de los alumnos.

Palabras clave: laboratorios virtuales y remotos, integración tecnológica, habilidades docentes.

Introducción

En los últimos años se han realizado numerosos trabajos tendientes a lograr una implementación adecuada de los llamados "Laboratorios Remotos" (LR). Estos esfuerzos han generado un conjunto de herramientas destinadas a fortalecer el concepto de Educación a Distancia (EAD), proveyéndoles a los alumnos, principalmente de las carreras de Ingeniería, los medios necesarios para encarar la realización de prácticas de laboratorio sin asistencia presencial a las aulas. Los resultados obtenidos con estas herramientas no solo han sido de gran impacto sobre el alumnado y los docentes [1], sino que han redituado en un mejor aprovechamiento de los laboratorios, ya que por medio del acceso web, estas facilidades se han podido utilizar fuera de los horarios normales de dictado de clases. De la misma forma se ha ocasionado una reducción de los costos involucrados en las tareas de mantenimiento del equipamiento asociado debido a la ausencia de manipulación física del mismo por parte de personal no necesariamente entrenado para su uso.

Sin embargo, desde el punto de vista de los sistemas de información, se ha apreciado una suerte de "divorcio" entre los sistemas de gestión de aprendizaje (LMS) utilizados para la gestión de los contenidos didácticos, incluyendo evaluaciones, y el aprendizaje teórico/práctico derivado del uso de los recursos de los laboratorios. Efectivamente, ambas herramientas de aprendizaje han seguido rutas sin un punto de contacto en común.

Si bien existen LMS específicamente orientados a la gestión de los laboratorios remotos y aprendizaje colaborativo [2], los mismos no guardan relación alguna con otros paquetes de software LMS de gran difusión y uso tradicional en diversas instituciones, tal como el caso de Moodle. Claramente, este

proceder ha traído algunas complicaciones derivadas del uso simultáneo de sistemas no compatibles ni enlazables entre sí, y que es necesario mantener sincronizados si se pretende dotar de coherencia al conjunto de ambos recursos.

Otro factor de impacto, derivado de la ausencia de integración, ha sido dado por el uso inadecuado de los denominados "Objetos de Aprendizaje" (OA). Los OA son entidades, codificadas según un formato estándar, que comprenden todos los contenidos necesarios para completar el aprendizaje de alguna unidad de conocimiento disponible en una asignatura. Estos contenidos abarcan no solo el desarrollo teórico de la unidad de conocimiento, sino que también incluyen glosarios de términos, autoevaluaciones, ejercitación de los conceptos adquiridos y todo aquel material que sea necesario proveer al alumno para garantizar el correcto desarrollo del proceso de enseñanza-aprendizaje. Dentro de este listado de materiales, paradójicamente, no figuran las prácticas de laboratorio, que han probado ser un instrumento de capital importancia para la formación académica de los alumnos de las ciencias relacionadas.

Si bien existen varios estándares para la creación y distribución de OA, tales como SCORM e IMS, ninguno de ellos prevee la inclusión de prácticas de laboratorio entre las actividades a realizar a lo largo del desarrollo de un OA, lo que en cierta medida potencia los problemas antes mencionados.

Por otra parte, varios investigadores [5][6] han propuesto sistemas de información que permiten gestionar las actividades en los laboratorios remotos y desarrollar prácticas remotas en los mismos, entre las cuales han tenido buena aceptación aquellos sistemas aplicables a electricidad, electrónica y control automático. Muchos de estos sistemas han cobrado relevancia en virtud de su facilidad de uso y de su depurada interfaz de usuario, pero sin embargo se mantienen como unidades separadas de los LMS tradicionales o exigen la migración de los contenidos a este nuevo sistema.

Por último, existe una marcada separación entre la implementación real de los mecanismos de control aplicados a las plantas físicas y las herramientas empleadas en las aulas [3] [4]. Específicamente, para lograr el controlador de una planta se suelen utilizar productos desarrollados para ello, tales como el paquete de software LabView® de National Instruments que a pesar de su muy destacada calidad, no se encuentra vinculado con las herramientas utilizadas por docentes y alumnos para el desarrollo y aprendizaje de los contenidos. Un claro ejemplo lo constituye Matlab® de MathWorks. Virtualmente no hay tema de Control Automático que no pueda modelarse y simularse por medio de este paquete de software, y de hecho, se lo utiliza masivamente para la enseñanza del Control Automático en los claustros universitarios. Sin embargo, a la hora de implementar un lazo de control sobre una planta real, normalmente es dejado de lado y reemplazado por otras utilidades, aún cuando este software posee mecanismos que permiten realizar y validar esta implementación sobre la misma herramienta, o generar código ejecutable capaz de operar sobre otras plataformas destinadas específicamente al control de procesos. Es probable que el acceso a las características de esta solución se encuentre algo oscurecida por la ausencia de documentación clara, pero no por ello es necesario dejarla de lado para profundizar en otras opciones, invirtiendo tiempo y dinero, cuando la mayor experiencia ya se encuentra disponible sobre ella. Logrando esto, será posible que los propios docentes diseñen e implementen los controladores a evaluar, y que los alumnos sean capaces de visualizar el desarrollo, ambos empleando para ello la misma herramienta que ya conocen y dominan.

Restricciones para el diseño

Analizado desde el punto de vista de las tecnologías actualmente disponibles, es fácil

comprender que el diseño de una plataforma para la implementación de laboratorios remotos se reduce, básicamente, a la integración de un conjunto de componentes tecnológicos ya disponibles, y al diseño y aplicación de algunos componentes de software adicionales que ofician de enlace entre los anteriores. Por esta razón, el foco del diseño debe estar centrado en una serie de problemáticas anexas, cuya solución debe ser asistida tecnológicamente, tales como las arriba enunciadas.

A fin de proporcionar un solución unificada a estos problemas, se fijaron un conjunto de restricciones que debían ser cumplidas estrictamente a lo largo del desarrollo. Ellas fueron:

1. Diseñar una arquitectura abierta y flexible para la implementación de laboratorios remotos de Control Automático (LRCA), independiente del tipo de planta remota a implementar y con la capacidad de vincularse con un gestor LMS, haciendo uso intensivo de los estándares disponibles.
2. Diseñar un plug-in de Moodle que sea capaz de interactuar con todos los recursos de la plataforma LMS para gestionar en forma eficiente el acceso a los recursos de los LRCA. Este plug-in también deberá ser capaz de interactuar con los OA, a fin de que los docentes puedan fijar las condiciones que deberán cumplir los alumnos para acceder a la realización de las prácticas de laboratorio [13]. Cabe aclarar que se eligió a Moodle como LMS, ya que es la plataforma en uso por la Universidad Nacional de San Juan (UNSJ) y por que es el LMS más difundido y utilizado en la actualidad.
3. Adoptar SCORM para la generación y distribución de los OA, en virtud de que es el estándar mas difundido y que dispone de la mayor cantidad de software de soporte.
4. Diseñar el controlador de la planta por medio de Matlab/Simulink con el apoyo de su Real Time Workshop (RTW), permitiendo de esta manera reutilizar la vasta experiencia de los docentes y alumnos con esta herramienta.
5. Utilizar la herramienta de simulación Easy Java Simulation (EJS) [14] como medio de interacción con el LRCA ya que es una excelente herramienta de simulación publicada como software libre.
6. Diseñar los módulos de software requeridos para enlazar el controlador generado con un sistema servidor capaz de publicar en Internet los medios de acceso al LRCA. Este sistema deberá proveer los servicios necesarios para interactuar con el laboratorio utilizando protocolos de comunicación estándares, de manera tal de permitir un acceso controlado y libre de restricciones tecnológicas severas.
7. Asegurar un mecanismo de interacción entre el LMS y el sistema del LRCA que sea independiente de ambos, con el objetivo de facilitar la migración a nuevas y/o distintas plataformas de manera no-traumática.
8. Utilizar una plataforma de hardware estándar para el sistema controlador de la planta, estando formado, en lo posible, por una o mas computadoras tipo PC equipadas con sistemas de adquisición de datos soportados por Matlab o extensiones adecuadas del mismo.
9. Utilizar productos de software libre en forma masiva, a fin de reducir de manera significativa el costo de la implementación de la solución propuesta.
10. Desarrollar software utilizando lenguajes de programación que generen código capaz de ejecutarse sin inconvenientes en múltiples plataformas de cómputo y acotando a lo estrictamente necesario la producción de código dependiente de la plataforma, lo que redundará en una futura migración no-traumática a otros entornos operativos.

Diseño de una arquitectura abierta y flexible

Teniendo en cuenta las premisas antes enumeradas, el primer paso consistió en el

diseño de una arquitectura informática multicapa capaz de proveer el soporte necesario para la implementación del LRCA.

En un extremo de esta arquitectura se encuentra el programa controlador de la planta de laboratorio, desarrollado en MATLAB y Simulink con apoyo del RTW. En el extremo opuesto se encuentra la aplicación EJS, proporcionando la interfaz de usuario y los medios de interactuar con el LRCA. Entre ambos extremos existe un conjunto de componentes de software que ofician de enlace entre el LRCA y el usuario.

Adicionalmente, se requiere un plug-in de Moodle con la capacidad de gestionar la administración de los laboratorios remotos a la usanza de los laboratorios presenciales, pero con información brindada por la interacción entre los alumnos y los OA que soportan los contenidos del espacio curricular en cuestión.

Plug-in de Moodle

Este componente de software fue el primero desarrollado en el marco de esta investigación sobre LRCA a fin de determinar la viabilidad de interactuar con la información gestionada por el LMS, incluyendo los OA.

Los resultados obtenidos con este módulo fueron muy promisorios, llegando al punto de autorizar en forma automática la realización de prácticas de laboratorio si el alumno ha cumplido con los requisitos fijados por los docentes a lo largo de los OA. Para mayores detalles, se remite al lector a la referencia [13].

Controlador de la Planta

Los programas controladores tienen la misión de recibir el valor de set-point que fije el usuario en forma remota, ejecutar el algoritmo adecuado, determinar las acciones de control a aplicar a la planta para que las variables controladas alcancen los set-point fijados y por último, operar en consecuencia sobre los actuadores de la planta bajo control. Esta tarea se repetirá a lo largo del tiempo a una cadencia determinada por el período de muestreo elegido según la dinámica de la planta en cuestión.

El desarrollo de este algoritmo de control, su simulación y su ejecución como controlador en tiempo-real es, probablemente, el ítem de mayor peso en el conjunto de las tareas a realizar. El concepto primordial era que el algoritmo fuera desarrollado y verificado por las personas con la mayor experiencia: los docentes, y luego fuera convertido en un componente ejecutable en tiempo real de la manera mas simple y expeditiva posible. De esta forma, no habría diferencia alguna entre los mecanismos de enseñanza y la aplicación de los conocimientos, y los resultados prácticos seguirían fielmente los valores de la simulación preliminar. Para lograrlo, se recurrió al empleo del paquete de software Simulink, ampliamente conocido y utilizado en todos los ambientes académicos, y que presenta un número importante de ventajas:

Matlab/Simulink permite el desarrollo “visual” de los algoritmos de control mediante un diagrama de bloques interconectados que incluye componentes tales como ganancias estáticas, sistemas de orden **n**, retardos, etc, de manera tal que la interconexión simboliza no solo las relaciones entre las entradas y salidas de los bloques, sino también la secuencia en las que se procesan tales entradas y salidas. Claramente puede observarse que este tipo de diseño “visual” es muy simple de realizar, ya que básicamente es igual al desarrollo del controlador según un diagrama en bloques del sistema, tal como sucede en las clases teóricas de Control Automático.

Este mecanismo de diseño asistido gráficamente permite construir diagramas (sistemas) de gran complejidad y pasar rápidamente a la etapa de **simulación** luego de tan solo haber dibujado el diagrama y haber validado las relaciones entre los bloques operativos. Nótese la presencia remarcada de la palabra “simulación”: esto es así por cuanto

Simulink es una herramienta de apoyo a Matlab para el diseño de sistemas simulados que serían extremadamente complejos de modelar matemáticamente, logrando así que el investigador pueda concentrarse rápidamente en los aspectos relevantes del problema a tratar, dejando de lado la complicación inherente a la codificación procedural del modelo a analizar.

Sin embargo, desde hace muchos años la empresa Mathworks, creadora de este producto, añadió a Matlab/Simulink la capacidad de incorporar y compilar código en lenguaje C de manera de agregar funcionalidades extra a los algoritmos a simular y que no estuvieran contempladas en el diseño original de Matlab. Poco tiempo después, le agregó la posibilidad de generar y compilar código en lenguaje C a partir de un modelo en Simulink, de forma tal de poder ejecutar ese modelo en código nativo de alguna plataforma computacional con el objetivo de poder independizarse de la presencia de Matlab para la ejecución del modelo en tiempo real, y poder así utilizar dicho modelo en aplicaciones reales, tal como la que ahora nos ocupa. En este contexto, Matlab provee un conjunto de plantillas llamadas Real Time Targets (RTT) orientadas a la producción de código destinado a ejecutarse sobre un conjunto de plataformas de cómputo conocidas y aceptadas en la industria. De igual manera, y a los fines de realizar la evaluación preliminar del algoritmo implementado, se provee un RTT denominado Real Time Windows Target (RTWT), específicamente diseñado para operar bajo el sistema operativo Microsoft Windows® y comunicarse con una instancia de Simulink para facilitar la interacción y supervisión del proceso en ejecución.

En el estudio preliminar de esta característica de Matlab/Simulink se encontraron varios inconvenientes propios de la naturaleza de estas soluciones:

- El RTWT solo puede comunicarse con una instancia de Matlab/Simulink, pero no proporciona medios de acceder a la información de entrada y salida del modelo desde otros programas externos a ellos, como es necesario para los LR. A esta restricción hay que agregarle la imposibilidad de enlazar el modelo, con módulos externos desarrollados en lenguaje C. Esta falencia deriva de que el código generado y compilado para ejecutarse en el entorno del RTWT puede acceder a escasos recursos del sistema operativo anfitrión en razón de que debe operar en modo kernel, ya que es ejecutado bajo el control de un device driver propio de Mathworks quien a su vez se ejecuta también en modo kernel para asegurar el comportamiento “hard real time” (HRT).
- Los RTT adicionales provistos con Matlab están orientados a plataformas de cómputo específicas, con soporte nativo de tiempo real y muy diferentes de lo que una computadora tipo PC ofrece, lo que requería la implementación o adquisición de hardware dedicado a este tipo de trabajo.

En la búsqueda de soluciones a este conflicto se encontró disponible, junto a la distribución de Simulink, un RTT de propósitos múltiples [7], denominado Generic Real Time Target (GRTT), destinado a servir como framework general para el desarrollo e implementación de targets de ejecución no contemplados en la implementación original provista con el producto. El análisis del código del GRTT permitió descubrir que con algunas modificaciones relativamente simples de realizar era factible implementar un target adecuado a las necesidades de los LRCA. Del mismo estudio surgieron un conjunto de necesidades que debían ser resueltas cuando se realizara la modificación del GRTT:

1. El programa resultante del uso del nuevo RTT derivado del GRTT (de ahora en mas VLRTT por Virtual Lab Real Time Target) debe ejecutarse en “modo usuario”, a fin de poder disponer de todas las facilidades provistas por el sistema operativo, lo que exige implementar de alguna manera el

comportamiento HRT necesario sin recurrir a la ejecución del programa en modo kernel.

2. Al operar en modo usuario, los sistemas operativos actuales virtualizan el hardware I/O de la computadora, por lo que es imposible acceder a los puertos físicos I/O si no es por medio de un device driver en modo kernel diseñado a tal efecto. Dado que en esta implementación de un LRCA se utilizó un dispositivo de hardware para vinculación con la planta a controlar diseñado como tesis de grado por alumnos del Instituto de Automática (INAUT), y que dicho hardware requería acceso libre a la interfaz del puerto paralelo de una computadora tipo PC, fué necesario implementar algún medio de liberar el acceso a un conjunto de direcciones I/O específicas para su uso por el hardware.

La solución a la primer necesidad consistió en el uso de un device driver en modo kernel capaz de generar notificaciones periódicas al programa de usuario que se comunicara con él. Estas notificaciones toman la forma de un evento que se genera, con resolución de microsegundos, cuando finaliza el tiempo que tiene asignado. El programa solicita este evento y espera su ocurrencia para poder continuar con sus tareas. Esta espera se realiza en un busy-loop hasta la aparición del evento, lo que en cierta medida garantiza una respuesta rápida a la ocurrencia del mismo, pero de ninguna manera asegura el comportamiento HRT necesario. Este último se aproxima combinando la operación del busy-loop del programa controlador con la ejecución de dicho programa como una tarea de alta prioridad para el scheduler del sistema operativo. De esta manera se minimizan las latencias propias de un sistema operativo no-real-time y si bien no se garantiza el comportamiento HRT, se logra una aproximación razonablemente satisfactoria en términos de la dinámica de las plantas típicas a controlar en el laboratorio. Es importante destacar que el device driver utilizado forma parte de los ejemplos provistos por Microsoft en su Driver Development Kit (DDK) [8] y uno de cuyos modos de operación es el utilizado en esta solución.

La segunda necesidad fue resuelta mediante el uso de un producto de software libre destinado a facilitar el acceso a los puertos I/O en modo usuario bajo el sistema operativo Windows. Este producto se denomina PortTalk [9] y está formado por un device driver en modo kernel y un pequeño conjunto de invocaciones ioctrl para comunicarse con él. Estas invocaciones no son un medio para acceder a los puertos I/O, sino que se utilizan para liberar el acceso a un conjunto de direcciones I/O en forma permanente mientras dura la ejecución del programa que las invoca, quien puede enviar y recibir datos I/O usando las primitivas I/O clásicas del lenguaje C (_outp e _inp) sin ningún tipo de restricciones sobre el espacio de direcciones I/O liberado.

Debe quedar claro que esta última solución fue implementada a raíz de la necesidad de utilizar un componente de hardware previamente desarrollado en el INAUT, pero de ninguna manera es necesario utilizarla si se emplea algunos de los dispositivos I/O soportados nativamente por Matlab/Simulink. De todas formas, su operación se encuentra integrada en el VLRTT y funciona solo en caso de recibir la indicación de hacerlo, por lo que este target puede ser utilizado tanto para operar conjuntamente con dispositivos I/O heredados como con dispositivos soportados en forma nativa.

Enlace de VLRTT con programas externos

Una de las premisas básicas en el diseño del target VLRTT fue que este contemplara la posibilidad de recibir y transmitir información desde y hacia programas externos. Esta consideración es de capital importancia en el contexto de un LRCA ya que el laboratorio remoto debe proporcionar la capacidad de fijar las referencias y set-point así como también monitorear la evolución de las variables

controladas a lo largo de la duración del experimento remoto.

El GRT incluye soporte para el trabajo en modo “remoto” utilizando comunicaciones en red bajo protocolo TCP/IP, el cual se elige y agrega como una propiedad del modelo Simulink antes de efectuar la generación y compilación del código nativo de la plataforma de cómputo. Sin embargo, este modo remoto está específicamente diseñado para que el modelo generado sea capaz de interactuar con Matlab/Simulink, de manera tal de poder observar el estado del modelo en cada momento utilizando las facilidades propias de este software.

Aún cuando era posible analizar el funcionamiento del protocolo de transferencia utilizado y desarrollar software capaz de comunicarse con el modelo utilizándolo, se decidió dejar de lado esta posibilidad a fin de independizarse de las necesidades propias de Simulink contempladas por este sistema de comunicación y así de lograr un mecanismo de transferencia de información mas simple y que pudiera ser integrado con facilidad con componentes de software externos al modelo. Es así que se eligió el uso de memoria compartida como medio de pasaje de datos entre el modelo y los programas externos. Si bien esta elección puede parecer antojadiza, su razón de ser se basa en que:

- Es un mecanismo de intercambio de información simple y eficiente.
- Está soportado en forma nativa por cualquier sistema operativo moderno, lo que facilita las tareas de migración a nuevos entornos de procesamiento en caso de ser necesario.

Dada esta elección, se desarrollaron un par de componentes de Simulink (MLabInNew y MLabOutNew) destinados a personificar un bloque de entradas, del cual el modelo leerá los valores que asumirán los set-point deseados; y otro bloque de salidas, en el cual el modelo escribirá los valores de las variables supervisadas. Cada uno de estos bloques tiene un número variable de salidas o entradas, configurables por el creador del controlador siguiendo la misma filosofía usada por los componentes **mux** y **demux** propios de Simulink, con la salvedad de que los orígenes y destinos de datos que salen y llegan a ellos son completamente externos al algoritmo desarrollado.

La creación de estos bloques en Simulink es una tarea relativamente simple y que solo debe hacerse una única vez para luego reutilizarlos en tantos desarrollos como sean necesarios. Para ello solo se requiere escribir el código en lenguaje C que realice las tareas de transferencia por medio de funciones diseñadas a tal efecto, respetando las especificaciones de Simulink. Para que estos bloques puedan ser incorporados al código real-time, es necesario realizar una etapa adicional que consiste en diseñar un archivo de metacódigo destinado al preprocesador del Target Language Compiler (TLC) para que este pueda expandir inline el código C utilizado.

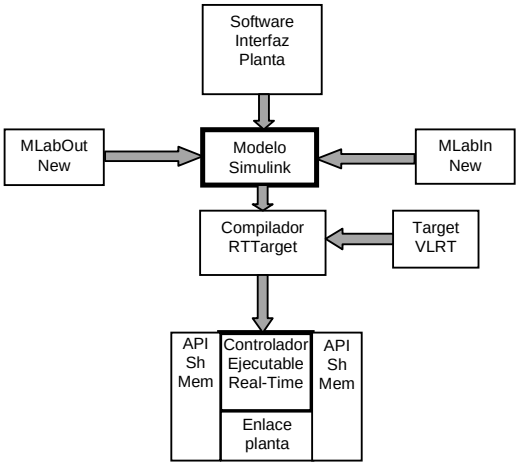

Figura 1: Construcción de un controlador real-time usando Simulink.

La Fig.1 muestra el proceso de construcción de un programa controlador usando Simulink y los componentes de apoyo desarrollados.

Otros componentes arquitecturales

Siguiendo los lineamientos antes expuestos, debe ser claro que resulta completamente factible diseñar el controlador de una planta de laboratorio utilizando Matlab/Simulink, a la vez que se integran en dicho controlador los mecanismos necesarios para permitir su comunicación con programas externos al mismo. Esta comunicación permite el acceso a los datos, expuestos por medio de las interfaces de memoria compartida, por parte de otros módulos de software que se encargarán de intercambiar información con los subsistemas que brindarán al usuario la interacción con el laboratorio remoto.

En el diseño presentado, se han implementado estos módulos usando la tecnología Java para lo cual se ha desarrollado un servidor de acceso a memoria compartida que expone tres interfaces remotas vía servicios RMI [10]. Estas interfaces son RemoteMatlabWriter (escritura en memoria compartida), RemoteMatlabReader (lectura de memoria compartida. Ambas interfases envuelven clases que utilizan la interfaz JNI [11], ya que el acceso a memoria compartida requiere el soporte de métodos nativos. El uso de RMI simplifica el acceso remoto a las interfaces exportadas y facilita su uso desde otras computadoras de manera completamente similar a la de objetos disponibles localmente.

Dado que RMI no es un protocolo de comunicaciones apto para ser utilizado directamente sobre Internet, ya que demanda complicadas configuraciones de los cortafuegos de protección perimetral de las redes y provoca vulnerabilidades, se hace necesario el uso de algún otro método de comunicación con la interfaz cliente del LR. En virtud de la necesidad de intercambiar datos básicamente numéricos entre el cliente y el LRCA, se ha optado por el uso de Web Services (WS) [12] como metodología de acceso a la información.

El uso de WS en esta aplicación es bastante simple ya que solo es necesario disponer de un reducido número de métodos remotos destinados enviar datos, recibir datos y ejecutar el controlador, y estos métodos son los mismos que son expuestos por las interfaces JAVA exportadas vía RMI. El uso de WS supone el acceso al laboratorio remoto a través de los servicios y protocolos de un servidor web, por lo que la complicada comunicación RMI se ha transformado en un simple intercambio de texto SOAP sobre HTTP, lo que no implica ninguna complicación adicional en los sistemas de seguridad perimetral de la red.

Por último, el subsistema cliente del LRCA está constituido por la aplicación Easy Java Simulation (EJS), modificada de manera adecuada para permitir el acceso a los servicios del LRCA operando como cliente de los WS implementados.

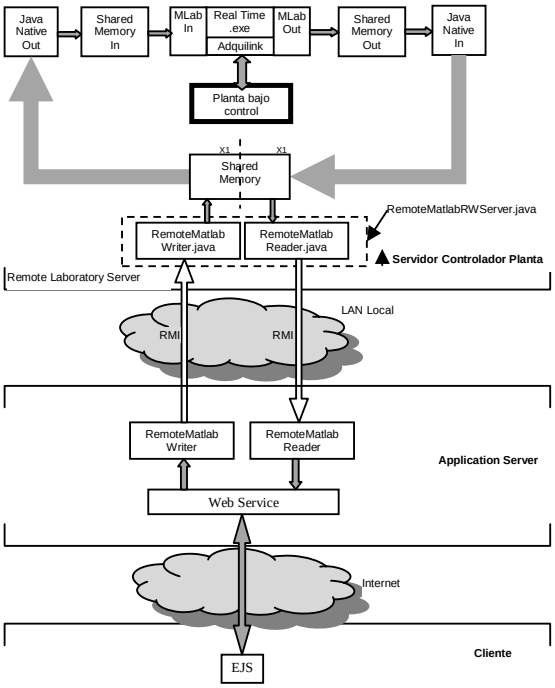

Figura 2: Arquitectura cliente-servidor completa

En la Fig. 2 puede observarse que la arquitectura propuesta consta de un programa controlador destinado a regir el

funcionamiento de la planta, el cual es generado por el subsistema de tiempo-real de Simulink en base a una plantilla diseñada a tal efecto (VLRTT). A este programa se adosan un par de bloques de Simulink cuya misión es brindar acceso a un área de memoria compartida, en la que programas externos escribirán o leerán los datos que resulten de la interacción remota del cliente. Para permitir esta interacción remota, se han diseñado un conjunto de clases e interfaces JAVA con capacidad de acceder a este área compartida, y cuyos métodos han sido exportados vía RMI para facilitar su uso remoto. Con el objetivo de simplificar el acceso a estos datos por medio de Internet, los accesos RMI han sido envueltos por medio de web-services, de forma tal que el application server que ejecuta estos WS opera como cliente del servicio RMI para recuperar la información de la planta real y como servidor de WS, traduciéndola a mensajes SOAP para que pueda viajar montada en paquetes HTTP desde y hacia la computadora del usuario final donde se ejecuta la aplicación EJS. Esta última es la encargada de mostrar los diagramas representativos de la planta remota y permitirle al usuario modificar aquellos valores que así estén previstos.

Evaluación preliminar

En el INAUT se diseñó y construyó una planta piloto destinada a la implementación de un laboratorio remoto y es sobre la cual se han realizado las evaluaciones preliminares de la arquitectura propuesta. En la Fig. 3 se puede apreciar la planta en cuestión, que es un sistema de segundo orden compuesta por dos tanques vinculados entre sí y con un depósito desde donde se provee el agua que llena los tanques hasta los límites fijados por el cliente remoto. El agua de este conjunto de tanques es recirculada para mantener completa libertad en la operación de la planta.

El tiempo de muestreo requerido por esta planta es del orden de los 100 milisegundos, habiéndose medido un jitter asociado de aproximadamente 90 microsegundos en promedio con picos máximos de 140 microsegundos. Este jitter es inferior al 0.2% del tiempo de muestreo total y se logró de forma relativamente sencilla con una configuración adecuada de la prioridad de ejecución del programa controlador sobre una computadora PC con procesador dual-core. Si bien estos son resultados completamente preliminares, resultan muy prometedores dada la simplicidad de la técnica de muestreo utilizada.

Figura 3: Vista de la planta piloto desarrollada.

Conclusiones

En virtud de lo expuesto, es posible manifestar que se han cumplido las premisas planteadas al inicio de este trabajo y se dispone en la actualidad de una arquitectura para la implementación de un LRCA, dotada de la apertura necesaria para implementar cualquier tipo de laboratorio remoto, modificando solo aquellas partes de no se ajusten a las nuevas demandas surgidas.

Trabajo Futuro

Al momento de la redacción de este trabajo las tareas se encuentran centradas en un rediseño de los WS y sus módulos clientes para compatibilizar ambos con el modelo

subyacente en EJS. Por otra parte, el plug-in de Moodle sufrirá algunos cambios a fin de dotarlo de la capacidad de otorgar a cada alumno un ticket electrónico que lo habilite para identificarse frente al sistema y hacer uso del turno de práctica reservado. Este ticket tendrá incorporado el concepto de Firma Digital con el objetivo de evitar adulteraciones y permitir su validación por el sistema del LRCA.

También está previsto realizar un desarrollo análogo sobre plataforma Linux, ya que la presencia de distribuciones y patches con soporte real-time (tal como RTAI) facilitarían notablemente el proceso y reducirían sustancialmente los costos en licencias de software.

Agradecimientos

Los autores agradecen a la Universidad Nacional de San Juan por la concesión y financiamiento del proyecto de investigación I-845.

Referencias

1. Remote Laboratorios versus Virtual and Real Laboratorios – Z. Nedic, J. Machota, A. Nafalski – Session T3E-1 – 33rd ASEE/IEEE Frontiers en Education Conference, 2003, Boulder, Colorado.
2. A framework for sustaining the continuity of interaction in Web-based learning environment for engineering education – Anh Vu Nguyen, Yassin Rekik, Denis Gillet – Proceedings of ED-MEDIA 2005 conference.
3. El sistema de tres tanques: Un laboratorio virtual y remoto usando Easy Java Simulations – N. Duro, H. Vargas, R. Dormido, S. Dormido, J. Sánchez – Actas de IV Jornadas de Enseñanza a través de Internet/Web de la Ingeniería de Sistemas y Automática, EIWISA 2005, pp.51 – 58.
4. Laboratorios Virtuales Remotos Usando Easy Java Simulations y Simulink – G. Farías, F. Esquembre, J. Sanchez, S. Dormido – XXVII Jornadas de Automática, Almería, 2006, pp. 926 – 933.
5. Efficient Integration Of Real-Time Hardware and Web Based Services Into MATLAB – S. Müller, H. Waller – 11th European Simulation Symposium and Exhibition, Castle, Erlangen-Nuremberg, Germany, 1989.
6. A Java Based Remote Laboratory for Distance Learning – S. Hsu, B. Alhalabi, M. Ilyas – ICEE 2000, Taiwan.
7. Ayuda de MATLAB/Simulink (disponible como parte de la instalación del producto).
8. Microsoft Windows Server 2003 SP1 Driver Development Kit documentation – "Event" Device Driver.
9. PortTalk: A Windows NT I/O Port Device Driver – www.beyondlogic.org/porttalk/porttalk.htm
10. JAVA Tutorial – Trail: RMI – http://java.sun.com/docs/books/tutorial/rmi/index.html
11. JAVA Tutorial and Code camps – Trail: JNI – http://java.sun.com/developer/onlineTraining/Programming/JDCBook/jni.html
12. Eclipse tutorials - Creating Bottom Up Web Services via Apache Axis2 – http://www.eclipse.org/webtools/community/tutorials/BottomUpAxis2WebService/bu_tutorial.html
13. Un enfoque integrado para las prácticas de laboratorio en la educación a distancia – M. I. Masanet, E. Zavalla, A. Fernandez – XV Congreso Argentino en Ciencias de la Computación, CACIC 2009, Jujuy, Argentina.
14. EJS website – http://www.um.es/fem/Ejs/