



Máster en Ingeniería de Sistemas y Control
Trabajo de Fin de Máster

# Generación y despliegue automático de laboratorios remotos con EjsS y Node.js

Alumno: Iñigo Aizpuru Rueda
Directores: Eva Besada Portas y José Antonio López Orozco
Curso 2017-2018
Convocatoria de defensa: Septiembre 2018

Máster en Ingeniería de Sistemas y Control
Trabajo de Fin de Máster

# Generación y despliegue automático de laboratorios remotos con EjsS y Node.js

Alumno: Iñigo Aizpuru Rueda
Directores: Eva Besada Portas y José Antonio López Orozco

# Autorización

Autorizo a la Universidad Complutense de Madrid (UCM) y a la Universidad Nacional de Educación a Distancia (UNED) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Iñigo Aizpuru Rueda

# Agradecimientos

A Eva Besada Portas y José Antonio López Orozco, directores del proyecto fin de máster por su inestimable ayuda y tiempo que me han dedicado, sin el cual no hubiera podido finalizar este trabajo.

A Francisco Esquembre, por la aportación de las excelentes ideas que cambiaron totalmente el enfoque del trabajo y que han guiado su desarrollo, suponiendo una gran parte él.

A Mónica, mi mujer, por su paciencia y porque tras superar una grave enfermedad, ha sabido mantenerse fuerte y ha seguido dándome ánimos y apoyo en todo momento.

A mis hijos, Alejandro, Sara y Blanca, por comprenderme en todos los momentos en los que no he podido estar con ellos.

Y a mi madre y suegros por encargarse de los niños en los momentos en los que nosotros no podíamos y en recuerdo muy especial a la memoria de mi padre, al que le hubiera gustado ver este trabajo terminado.

# Resumen

El presente Proyecto Fin de Máster tiene como principal objetivo automatizar el despliegue de laboratorios remotos en la Universidad Complutense de Madrid (UCM) utilizando para ello Easy Java/JavaScript Simulations (EjsS) en la parte de cliente y Node.js en el servidor.

Para su realización se ha construido una infraestructura de *Plugins* sobre EjsS que permite extender su funcionalidad más allá de lo que su sistema de extensiones estándar permitía. Esta misma infraestructura se ha utilizado para crear un *Plugin* con varias extensiones para lograr nuestro objetivo de forma cómoda e integrada con el uso de una única herramienta.

También se han implementado una serie de facilidades a la propia creación de interfaces de laboratorio sobre EjsS que aportan robustez al diseño y aceleran el proceso de creación y despliegue de nuevos laboratorios, así como durante el mantenimiento. Estas facilidades incluyen nuevos elementos visuales compuestos e incluso elementos que permiten probar nuevas interfaces de laboratorio sin dejarlo fuera de servicio.

Adicionalmente, se ha implementado un mecanismo que permite tanto a los diseñadores del laboratorio como a los usuarios de los mismos, codificar y desplegar su propio código de controlador de forma remota desde la misma herramienta, ampliando los horizontes de los experimentos que se pueden realizar sobre ellos.

Las pruebas de despliegue automático se han realizado tanto sobre un laboratorio virtual creado ad hoc como sobre un laboratorio remoto real existente en la UCM destinado a la realización de prácticas de control de diferentes titulaciones.

**Palabras clave**: Laboratorios Remotos, EjsS, Node.js

# Tabla de Contenido

# Tabla de Ilustraciones

# 1 Introducción

En la actualidad, nos encontramos en un estado en el que la ciencia avanza a pasos agigantados y esto es posible gracias al gran esfuerzo de investigación y la gran cantidad de ensayos que se realizan diariamente en laboratorios de todo el mundo. Al pensar en un laboratorio nos pueden venir a la cabeza multitud de escenarios posibles enmarcados dentro de diferentes ámbitos como pueden ser el laboral, militar, docente, etc. A pesar de que existen laboratorios con un sin fin de objetivos, en el presente trabajo nos centraremos en aquellos enmarcados en el ámbito educativo.

En un entorno docente universitario es habitual disponer de una serie de laboratorios en lo que los alumnos pueden interactuar físicamente con los elementos que lo componen, pero esta interacción se ve restringida a los horarios en los que dura la clase práctica, limitando mucho el tiempo en el que los alumnos pueden experimentar por sí mismos los conceptos vistos durante las clases teóricas. Una vez finalizada la clase práctica, el alumno no podrá hacer uso del laboratorio para realizar nuevos experimentos que le permitan ahondar en la materia y afianzar sus conocimientos.

El avance de las nuevas tecnologías ha posibilitado ir un paso más allá en las prácticas, permitiendo a los alumnos la interacción remota con los laboratorios ubicados en las instalaciones de la universidad desde una ubicación externa mediante el uso de un ordenador, teléfono inteligente o tableta desde cualquier lugar del mundo, naciendo así, los laboratorios remotos. Aunque para posibilitar tanto el acceso como la utilización del laboratorio se requiere de una infraestructura/tecnología que permita la conexión a internet, éste no es el único punto importante a tener en cuenta a la hora de diseñar un laboratorio remoto. El diseño de un laboratorio remoto debe contemplar muy bien la interfaz gráfica que se mostrará al alumno para que ésta sea lo más intuitiva y fácil de usar para que el alumno pueda aprovechar y sacar el máximo rendimiento al aprendizaje. El profesor/diseñador es el responsable de, mediante esta interfaz, delimitar las señales de entrada al laboratorio que el alumno podrá manipular así como de proteger el laboratorio de un uso que pueda dañarlo. Igualmente, la interfaz gráfica deberá mostrar de la manera más adecuada posible las salidas relevantes que se produzcan en el laboratorio, adaptadas a cada ocasión, siempre para ayudar al alumno a comprender mejor la materia de estudio.

## 1.1 Necesidad de Automatizar

Como se ha comentado, existe una extensa variedad de laboratorios que se pueden preparar para ser controlados de forma remota. Aunque muchos de ellos tendrán características similares, cada uno de ellos tendrá sus peculiaridades como pueden ser el número y tipo de entradas y salidas, parámetros de configuración, numero y tipo de controladores y un largo etcétera.

A la hora de convertir un laboratorio presencial en remoto, siempre se puede tratar de reutilizar en la medida de lo posible, elementos comunes a otros ya existentes, ahorrando tiempo y costes en su implantación. En muchos casos, existirán elementos que se puedan copiar de un laboratorio a otro con mínimas adaptaciones, pero la cosa cambia en el caso de las características específicas de cada uno. Cada elemento diferente necesitará una configuración y presentación específica que requerirá de una inversión de tiempo más o menos significativa.

Además, un profesor que quisiera poner en marcha un laboratorio remoto tendría que dominar muchas tecnologías para lograrlo y se vería sobrecargado con un trabajo añadido que resulta costoso, haciendo que la puesta en producción del mismo suponga una inversión de tiempo considerable. Este trabajo añadido abarca desde el diseño de una buena interfaz, la preparación de un servidor y de la plataforma de comunicaciones, la depuración y corrección de los errores que

aparezcan, e incluso el mantenimiento del propio laboratorio cuando se realizan modificaciones en el mismo.

Todas estas razones provocan que haya ocasiones en las que laboratorios que podrían ser perfectamente susceptibles de ser convertidos en remotos se queden en laboratorios presenciales, ya que los profesores no siempre dispondrán de este tiempo extra para implementarlos y mantenerlos.

Esto evidencia la necesidad de facilitar al máximo posible la creación y despliegue de un laboratorio remoto al equipo docente con el fin de mejorar la transmisión de conocimiento de diversas materias por parte del profesorado hacia el alumnado, de una manera más práctica. Una forma de aliviar esa necesidad comienza por dotar al profesor de los medios necesarios para facilitar la creación de laboratorios remotos, reduciendo los tiempos de creación y puesta en marcha de los mismos. Esto se consigue tanto mediante la creación de nuevas herramientas que aceleren la creación de interfaces con laboratorios remotos como mediante la automatización del despliegue de los mismos.

Esta automatización no resulta una tarea sencilla ya que además de la variedad de tipos de laboratorio, existe el factor de que diferente información puede ser manejada y presentada de muy diversas formas. Imaginemos por ejemplo un laboratorio de control muy simple en el que el objetivo es demostrar los efectos de diferentes tipos de controladores sobre una variable controlada. En este caso, se controlará la temperatura de un fluido circulante por un tubo, controlando una fuente de calor con un regulador preparado para funcionar en modo P, PI, PD y PID.

Además de construir el laboratorio en sí, el profesor debería configurar la infraestructura de comunicaciones para habilitar su uso remoto definiendo los parámetros que el alumno podrá manipular, como los parámetros del controlador de la fuente de calor y los valores que le serán devueltos como el tiempo y la temperatura actual. Este proceso será manual y dependerá exclusivamente del criterio del profesor. Una vez definidos, el profesor deberá decidir de qué modo podrá el alumno interactuar con el laboratorio.

Para permitir al alumno modificar los parámetros del controlador, un profesor podría decantarse por permitir al alumno introducirlos libremente en cajas de texto mientras que otro lo haría mediante el uso de barras de desplazamiento con límites superior e inferior para cada uno de los valores a modificar.

En el caso de la temperatura, un profesor podría considerar que la manera óptima de presentarla es mediante una gráfica temporal mientras que otro podría dar otro enfoque y querer presentarlo mediante valores numéricos que representen los valores máximo, mínimo y actual alcanzados durante el ensayo sin dar ninguna importancia al tiempo. Otro profesor podría simplemente querer probar varias posibilidades antes de decantarse por una opción. Tanta diversidad dificulta la automatización de la creación de interfaces con laboratorios remotos en base a entradas y salidas, pero enfocándolo del modo adecuado se puede facilitar enormemente esta tarea manual.

Finalizada la preparación se procederá a la puesta en producción del laboratorio remoto. El uso de la tecnología apropiada para lograr este objetivo es crucial y de ella dependerán los pasos a seguir para su despliegue y de la facilidad del proceso. Una buena elección de la tecnología podría permitir el despliegue de un nuevo laboratorio remoto de forma automática.

## 1.2 Alcance del Trabajo y Objetivos

Para la realización de este trabajo se ha estudiado el caso concreto del modo en el que se implementan los laboratorios remotos en la Facultad de Ciencias Físicas de la Universidad Complutense de Madrid (UCM).

Por una parte, la UCM dispone de un conjunto de laboratorios accesibles por los alumnos de forma remota utilizando sus ordenadores personales, tabletas e incluso teléfonos inteligentes. El acceso a estos laboratorios se ha logrado mediante el uso de tecnología WEB, implementando en sus servidores de laboratorio una aplicación Node.js que interactúa con el software del laboratorio, permitiendo su control remoto. La herramienta utilizada para la creación de las interfaces gráficas de sus laboratorios remotos es Easy Java/JavaScript Simulations (EjsS), también utilizada por otras universidades como la Universidad Nacional de Educación a Distancia (UNED).

Para la creación de los nuevos laboratorios, el procedimiento habitual es partir de un laboratorio ya en funcionamiento y adaptarlo para las características específicas del nuevo laboratorio. Este proceso, a pesar de que ofrece una buena base de partida, requiere un esfuerzo importante por parte de los profesores que va creciendo a medida que crece la complejidad de los laboratorios. A medida que el número de elementos gráficos aumenta, también aumenta la posibilidad de cometer algún error y estos errores no siempre serán fáciles de depurar. Esto puede llevar al profesor incluso a desestimar la idea de hacer el laboratorio accesible de forma remota. Debido a esta dificultad, se plantea como primer objetivo el facilitar en la medida de lo posible la creación y mantenimiento de las interfaces gráficas de los laboratorios remotos.

Una vez creado el laboratorio, el profesor debe instalarlo en el servidor siguiendo para ello un procedimiento bastante mecánico pero manual y en este punto también se ha visto una interesante vía de mejora que nos lleva a nuestro siguiente objetivo que es precisamente lograr automatizar totalmente este proceso de despliegue de los laboratorios remotos.

Por otra parte, los laboratorios remotos de la UCM no están formados únicamente por la interfaz gráfica y la aplicación Node.js sino también por el software controlador del laboratorio. Este software es diferente en función de la plataforma elegida pudiendo ser controladores propios implementados en algún lenguaje de programación como puede ser C, u otro software de terceros como puede ser TwinCAT, LabVIEW o incluso MATLAB.

En el caso concreto de controladores propios, una de las preocupaciones del profesorado reside en que la implementación de los mismos la realizan en diferentes equipos, dándose el caso en el que existen ocasiones en las que no están plenamente seguros del controlador exacto que está corriendo en un laboratorio concreto. Por ello, el tercer objetivo que se contemplará será el ofrecer un mecanismo que pueda ser utilizado por el profesorado para mantener identificado el software instalado en sus laboratorios para facilitar su mantenimiento.

Resumiendo, los objetivos propuestos para el presente trabajo serán:

- Facilitar la creación de nuevos interfaces para laboratorios remotos y el mantenimiento de los interfaces ya existentes.
- Automatizar los trabajos de despliegue de interfaces gráficos de laboratorios, eliminando por completo todo proceso manual.
- Aportar mayor control sobre el código del controlador desplegado en los laboratorios.

## 1.3 Estructura del documento

Este documento está estructurado en 6 capítulos y dos anexos que describirán en detalle el trabajo realizado, los problemas encontrados y las soluciones aportadas para lograr los objetivos aquí planteados.

En el capítulo 2 se describe de una forma genérica en qué consisten los laboratorios remotos, los diferentes tipos que existen y el modo en el que la UCM ha afrontado la implementación de los mismos. En este capítulo también se describirán las herramientas utilizadas en la actualizad.

En el capítulo 3 se describe qué es lo que la herramienta EjsS puede hacer, así como el modo en el que se ha extendido su funcionalidad para permitir a los desarrolladores implementar nuevas soluciones integradas en ella.

En el capítulo 4 se verá en detalle la solución implementada para la creación y automatización del despliegue de laboratorios remotos, utilizando las nuevas capacidades añadidas a EjsS.

El capítulo 5 está dedicado a exponer los resultados prácticos obtenidos mediante la utilización de los elementos descritos en los capítulos anteriores.

En el capítulo 6 se extraerán las conclusiones del trabajo realizado y se propondrán nuevos trabajos que se pueden realizar en un futuro, como continuación a este trabajo.

El anexo A describe brevemente el modo en el que se gestionan los *Plugins* en EjsS así como los elementos necesarios para su creación.

Finalmente, el anexo B aporta unas nociones importantes a tener en consideración a la hora de instalar las actualizaciones de EjsS.

# 2 Laboratorios Remotos

Los laboratorios remotos son aquellos que están conectados a internet y que ofrecen la posibilidad de ser operados a distancia. Este tipo de laboratorios pueden ser utilizados tanto con propósitos industriales como de investigación y están implantados en gran cantidad de instituciones, entre los que se encuentran universidades de todo el mundo. A este respecto es importante mencionar especialmente a la UNED cuya tecnología se ha convertido en un referente internacional [1].

Centrándonos en el ámbito educacional, este tipo de laboratorios proporciona una gran ventaja sobre los laboratorios puramente locales ya que ponen los equipos del laboratorio a disposición de estudiantes ubicados en cualquier parte del mundo. Además, al poder acceder a ellos a través de internet se eliminan las restricciones horarias de utilización al estar disponibles durante más tiempo de un modo desatendido.

Incluso ciñéndonos al ámbito educacional, existe una amplia variedad de tipos de laboratorio que sin duda podríamos categorizar de muchas maneras distintas y en distintos niveles, bien agrupándolos de un modo muy general por la disciplina objeto del laboratorio, o siendo un poco más concretos, en el campo que estudia e incluso podríamos llegar a categorizarlos por el tipo de elementos que forman parte del laboratorio. Por ejemplo, alguien podría categorizar un laboratorio como “laboratorio de robótica” mientras que otro podría categorizar el mismo como “laboratorio de vehículos autónomos”. Este capítulo no pretende ser exhaustivo con su clasificación ni pretende enumerar todos los tipo que existen, pero sí dar un enfoque que ayude a entender mejor el ámbito en el que se ha desarrollado este trabajo.

## 2.1 Tipos de laboratorios remotos

Como una primera aproximación y desde un punto de vista muy general, podemos diferenciar los laboratorios en base a si son:

- Laboratorios reales.
- Laboratorios virtuales.

Los laboratorios reales son aquellos que están formados por equipos físicos reales. Dependiendo del campo de estudio, estos equipos serán diferentes, así un laboratorio de robótica contará con algún tipo de robot como puede ser un brazo robótico, un robot bípedo o un vehículo autónomo mientras que un laboratorio de automática podría estar compuesto por una serie de tanques de agua, motores y válvulas de control.

Los laboratorios virtuales son aplicaciones software que simulan el funcionamiento de laboratorios reales permitiendo a los alumnos experimentar en un entorno seguro, ya que carece componentes reales que puedan sufrir cualquier tipo de contratiempo. Entre las ventajas que aportan este tipo de laboratorios se encuentra el poder simular el comportamiento de procesos complejos o de equipamiento que resultaría muy caro e inaccesible para una institución, únicamente mediante el uso de modelos matemáticos. La puesta en marcha de estos laboratorios resulta mucho más sencilla de conseguir, al igual que su mantenimiento o modificación. Por contra, el inconveniente que sufren estos laboratorios es que los modelos matemáticos utilizados en ellos suelen estar simplificados, por lo que los resultados obtenidos, aunque tiendan a asemejarse a la realidad y puedan ser considerados válidos, no serán completamente fieles a ella.

Los laboratorios remotos, también pueden ser reales o virtuales. Los laboratorios remotos reales son aquellos que conectan los equipos físicos a internet. Estos laboratorios a menudo incorporan elementos tales como webcams para ofrecer a sus usuarios la visión de lo que está ocurriendo para acercar la experiencia al lugar en donde se encuentra.

Cuando hablamos de laboratorios remotos podemos cometer el error de pensar en que nos referimos únicamente a laboratorios reales físicos a los cuales accedemos mediante una interfaz virtual, pero no tiene porqué ser siempre así. También existen laboratorios virtuales a los cuales se puede acceder de forma remota, convirtiéndolos así, en laboratorios remotos. En estos casos, la aplicación software que ejecuta la simulación incluye típicamente un servidor web a través del cuál se servirá el interfaz del laboratorio virtual a modo de página web.

En este punto conviene destacar que a pesar de los muchos beneficios que ofrecen los laboratorios remotos, la conversión de un laboratorio local a remoto no siempre será sencilla. Las razones pueden ser varias, en algún caso nos podríamos encontrar con que debido a la naturaleza del propio laboratorio, el experimento podría requerir la actuación manual de un operario, o podría ocurrir que el fabricante del equipo o los instrumentos que estamos utilizando no ofrecen interfaz alguno para su gestión remota, limitando su uso al laboratorio local. En cualquier caso, siempre se puede tratar de encontrar soluciones que permitan su uso remoto. Su viabilidad dependerá del coste asociado.

Los laboratorios remotos, tanto reales como virtuales, están presentes en casi todos los campos de la enseñanza. Veamos ahora algunos ejemplos de laboratorios remotos reales.

En un lugar privilegiado se encuentra la UNED, pionera en la implantación de laboratorios remotos, de los que actualmente dispone de una cantidad importante de ellos. Estos laboratorios se encuentran actualmente bajo su portal UNILabs, que es una red de Universidades, entre las que se encuentran la propia UNED y la Universidad Complutense de Madrid, que contribuyen a crear un conjunto de laboratorios remotos y virtuales que ponen en común para uso de sus profesores y alumnos [6]. Por citar algunos ejemplos, la red UNILabs cuenta con laboratorios de control de nivel en una planta de 4 tanques, control de un cuatrirrotor de tres grados de libertad, control de sistema de bola y plato, control de robots móviles o control no lineal del péndulo de Furuta entre otros muchos.

*Fig. 1. Laboratorio de 4 tanques [2]*

Buscando un poco fuera de nuestro ámbito, podemos encontrar que en la *Universidad de León* disponen de una serie de laboratorios remotos entre los que se encuentra un laboratorio que implementa el problema de control de 4 tanques propuesto por Karl Henrik Johansson en "The quadruple-tank process: a multivariable laboratory process with an adjustable zero" [3]. Este laboratorio, mostrado en la figura 1, es un laboratorio preparado para ser utilizado tanto en local como en remoto. Se trata de un laboratorio que permite tanto experimentos de control sencillo como de control multivariable avanzado y está compuesto por varios tanques de agua, bombas de agua, válvulas y sensores de nivel. El laboratorio además incorpora salvaguardas que garantizan la funcionalidad e integridad del sistema en su uso tanto local como remoto a través de internet a usuarios autorizados. Además de este laboratorio, la *Universidad de León* dispone de muchos otros laboratorios de diferentes campos de estudio como son el control de un brazo robótico ABB de 6 grados de libertad, la programación remota de PLCs o la interactuación con una célula clasificadora neumática entre otros [2].

Otro ejemplo corresponde a la *Universidad de Deusto* que pone a disposición de los alumnos un laboratorio en el que pueden programar remotamente el procesador PIC de un pequeño robot móvil. El robot dispone de, entre otras cosas, sensores infrarrojos que el alumno podrá utilizar. La audiencia objetivo de este laboratorio son los estudiantes de ingeniería en general y determinadas escuelas de educación secundaria. La *Universidad de Deusto* también ofrece otros laboratorios remotos en los que, entre otros, se puede desde programar una FPGA hasta controlar el alimento de los peces de un acuario o un pequeño submarino cuando se encuentra situado en su interior y su batería lo permite [4].

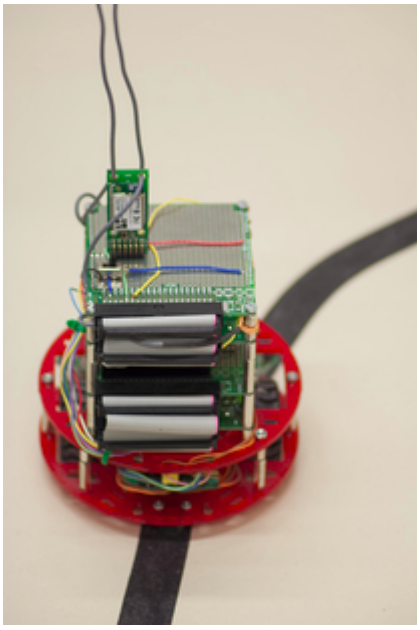

*Fig. 2. Pequeño Robot Móvil [4]*

Estas iniciativas no solo existen en España. Por poner un ejemplo, la Universidad de Chile también ofrece a sus alumnos la posibilidad de trabajar con laboratorios remotos [5].

Además de los ejemplos aquí mencionados, la lista de ejemplos prácticos que podemos encontrar es interminable y los conocimientos que aportan pertenecen a campos tales como control, automática, electrónica, robótica, domótica y muchos más. En el próximo apartado se describirán los elementos que típicamente conforman un laboratorio remoto.

## 2.2 Componentes de un Laboratorio Remoto Genérico

En el apartado anterior se han visto ejemplos de laboratorios remotos reales y se ha dado una noción de la gran variedad de ellos que existen. Los ejemplos citados se engloban todos ellos dentro del campo de la ingeniería y más concretamente en el área del control por ser éste el ámbito relevante para nuestro trabajo, aunque esto no significa que ésta sea la única disciplina que pueda implantarlos. Los laboratorios remotos a menudo suelen seguir el esquema mostrado en la figura 3.

En ellos, la funcionalidad se encuentra distribuida en tres capas que son:

- El front-end o interfaz de usuario.
- El back-end o motor del laboratorio.
- El middleware o capa intermediaria entre las dos anteriores.

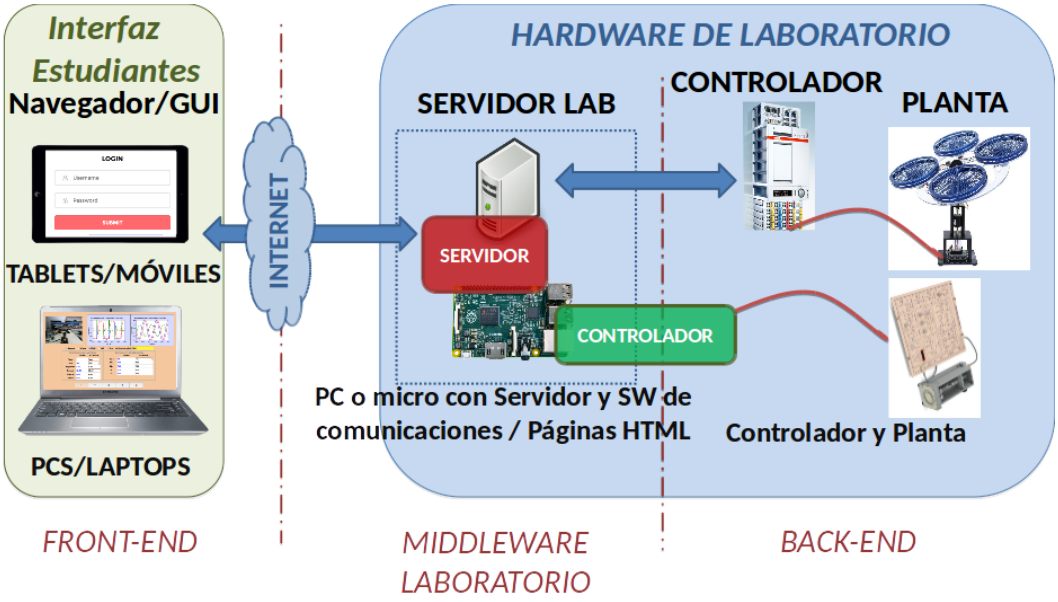


*Fig. 3. Esquema de laboratorios remotos genéricos*

En el back-end del laboratorio la aplicación controladora se encarga de leer las mediciones de las salidas de la planta y de realizar los cálculos de las señales de salida del controlador, cerrando así el bucle de retroalimentación de la planta.

En el otro extremo, la interfaz de usuario en el front-end permite a los alumnos:

- Autenticarse en el laboratorio remoto.
- Acceder a los datos del experimentos.
- Parametrizar el comportamiento del controlador y observar en tiempo real la evolución de las señales en planta.

Finalmente, el middleware gestiona el acceso al laboratorio y las comunicaciones entre la interfaz gráfica y la aplicación controladora del laboratorio.

## 2.3 Trabajos Realizados en la UCM

Como se ha visto, los laboratorios remotos son ya una realidad en muchas universidades entre las que se encuentra la Universidad Complutense de Madrid (UCM). Este trabajo tiene como punto de

partida los trabajos realizados dentro de la Facultad de Ciencias Físicas de la UCM, en la que ya disponen de varios laboratorios remotos de control. Allí se está llevando a cabo el proyecto denominado ReNoLabs [7], cuyo objetivo persigue la mejora continua de la infraestructura utilizada por la universidad para la implantación de laboratorios remotos y que implementa la arquitectura de tres capas mostrada anteriormente.

ReNoLabs implementa las comunicaciones entre el cliente y el servidor utilizando tecnologías web. En este caso, los servidores web están implementados con una aplicación Node.js, que es un entorno de desarrollo y ejecución basado en JavaScript, multiplataforma y de código abierto, cuya arquitectura basada en eventos de entrada y salida no bloqueantes permite la creación de aplicaciones de red en tiempo real, muy adecuada para estos laboratorios. La figura 4 muestra el primer nivel de acceso del alumno al laboratorio remoto. Esta aplicación Node.js corresponde al middleware y al front-end de la arquitectura descrita, lo que hace de éste, un elemento clave que hace de estos laboratorios una realidad. La interfaz de usuario está implementada en JavaScript y generada desde EjsS. En cuanto a las conexiones con el controlador de planta, se han desarrollado exitosamente interfaces con varios sistemas lo que permite conexiones remotas a laboratorios basados en TwinCat (ADS), LabView y aplicaciones propias programados en lenguajes como C, Java o Python [8, 9]

Las responsabilidades de la aplicación Node.js son:

- Gestionar el control de acceso restringiendo el uso del laboratorio remoto a usuarios autorizados.
- Atender las peticiones del cliente y servir las páginas solicitadas.
- Centralizar las comunicaciones entre el controlador y el cliente.
- Almacenar los resultados de los experimentos realizados por los alumnos para que puedan éstos revisarlos y estudiarlos posteriormente.

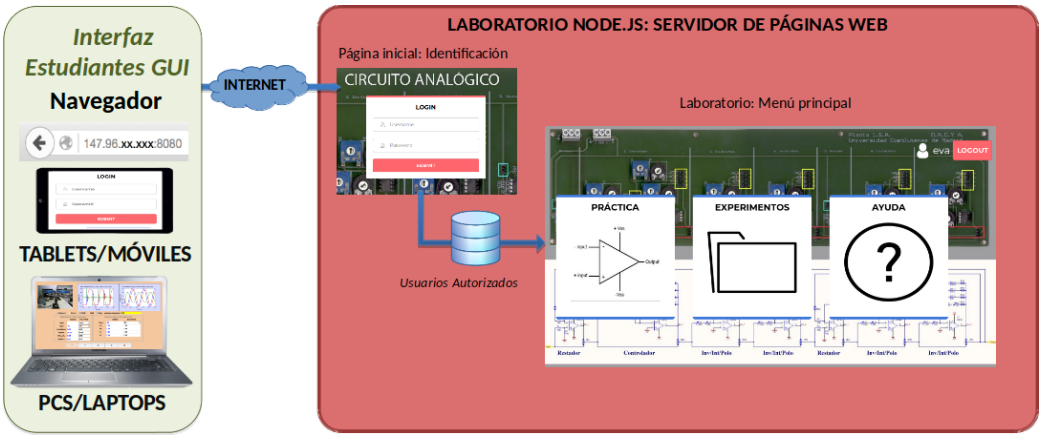


*Fig. 4. Acceso al Laboratorio Remoto con ReNoLabs*

Desde el menú principal de la aplicación, el alumno podrá acceder a la página web del controlador del laboratorio a través del icono correspondiente, en la cuál podrá realizar sus experimentos interactuando con la planta a través de la interfaz gráfica que el diseñador del laboratorio ha creado. La figura 5 muestra un ejemplo de una interfaz de laboratorio remoto.

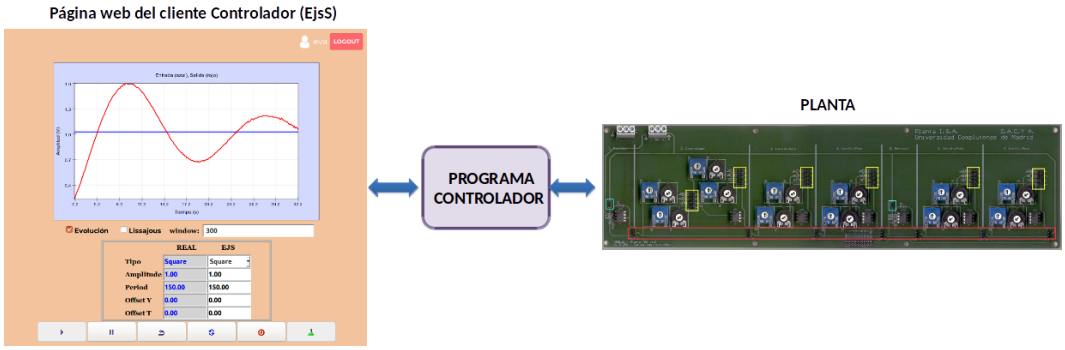


*Fig. 5. Ejemplo de Interfaz de Laboratorio Remoto con ReNoLabs*

A través de los otros dos enlaces que se presentan en la página principal, el alumno podrá acceder a una página de ayuda con la descripción del laboratorio y a otra página en la que podrá descargar los ficheros de texto con los datos de todas las prácticas que ha realizado.

En los laboratorios con aplicaciones propias, los elementos físicos del laboratorio están conectados a una *Raspberry Pi*, un computador compuesto de una placa de pequeño tamaño y bajo coste. El sistema operativo elegido para estos computadores es *Raspbian* sobre el que corre el software del controlador. Los controladores de estas aplicaciones están implementados utilizando principalmente C, pero podrían implementarse utilizando cualquier otro lenguaje de programación soportado por el sistema operativo como pueden ser C++ o Python. En ReNoLabs se están explorando estas posibilidades. En este tipo de laboratorios, tanto el servidor web con la aplicación Node.js como la aplicación que contiene el controlador se encuentran implementados en la propia Raspberry Pi comunicándose entre ellos mediante el uso de los canales de entrada y salida estándar (STDIO).

En los caso en los que se ha optado por los PLC de TwinCAT, los circuitos están conectados a un computador dedicado que hace de servidor del experimento que es con el que se comunica tanto para recibir las entradas del alumno como para devolver los datos del experimento. Este computador se comunica con el laboratorio mediante TwinCAT Automation Specification System. En este caso, el servidor de experimento se comunica con el servidor web mediante el módulo ADS que extiende la funcionalidad de Node.js.

A pesar de que los laboratorios descritos son laboratorios remotos reales, lo interesante de utilizar esta aplicación Node.js intermediaria es que, una vez definidas las entradas y salidas del experimento y el modo de comunicarse con el laboratorio, el hecho de que éste sea real o virtual será totalmente irrelevante para la aplicación. De este modo, se pueden integrar tanto laboratorios reales como virtuales en la misma plataforma.

La interfaz de usuario está implementada en JavaScript y HTML5 por lo que el acceso al laboratorio remoto se realiza a través de páginas web accesibles desde un explorador de internet. Esta tecnología permite a los alumnos acceder a las páginas de laboratorio utilizando tanto ordenadores personales, como dispositivos móviles, como teléfonos inteligentes o tabletas.

La generación de la GUI de estas páginas, así como su comportamiento, no resulta sencillo de implementar por lo que es esencial disponer de una buena herramienta para su creación. Existen muchas herramientas cuyo objetivo es la creación de páginas HTML, sin embargo, la herramienta seleccionada en ReNoLabs es una destinada a otro propósito. Esta herramienta no es otra que Easy Java / JavaScript Simulations (EjsS). Como se expondrá en el próximo capítulo, a pesar de que EjsS es una herramienta orientada a la creación de simulaciones, es también una potente herramienta para generar interfaces gráficos de modo muy sencillo debido a que dispone de una gran cantidad de elementos HTML que el usuario podrá utilizar simplemente arrastrándolos a una estructura

jerárquica de tipo árbol para dar forma a la página. ReNoLabs ha sabido sacarle partido a esta característica para la creación de las páginas web del laboratorio remoto.

A pesar de todo el trabajo realizado por el equipo de ReNoLabs, la creación, mantenimiento y despliegue de los laboratorios sigue un proceso manual que presenta los problemas típicos descritos en la introducción. Este proceso incluso requiere de modificaciones manuales en el código generado por EjsS, siendo un proceso costoso y posible generador de errores.

## 2.4 Herramientas para la Automatización

En un laboratorio se utilizan muchas tecnologías que el desarrollador del mismo debe conocer, tanto para implementar el laboratorio físico como para habilitarlo para su uso remoto. A la hora de realizar un estudio de las herramientas a utilizar para la generación automática del laboratorio remoto, se ha tenido en consideración la dificultad que conlleva el añadir nuevas tecnologías a un entorno conocido tanto desde el punto de vista técnico como de aprendizaje del desarrollador. Por ello y visto el éxito y el grado de avance de ReNoLabs, se tomó en una fase muy temprana la decisión de utilizarlo como base para el proyecto, reutilizando las tecnologías implantadas abordando la automatización sobre ellas.

Mientras ReNoLabs sigue su curso integrando nuevos elementos en los laboratorios remotos abriendo puertas a nuevos modos de implementar laboratorios remotos, el presente trabajo aporta a ReNoLabs la capacidad de poder automatizar el despliegue de los mismos con el mínimo esfuerzo y desde el mismo computador en donde se diseña el laboratorio.

En el apartado anterior se han nombrado a muy alto nivel algunos de los componentes software que utiliza ReNoLabs, a continuación se dará una descripción más detallada de estas herramientas y los módulos que conforman el cliente y el servidor. No se describirán los elementos de laboratorio ni las comunicaciones con ellos por ser algo muy específico de cada uno.

### 2.4.1 Herramientas de Servidor

El middleware del servidor del laboratorio está compuesto por una aplicación desarrollada sobre Node.js. Éste es un entorno de desarrollo y ejecución basado en JavaScript y que ofrece una gran cantidad de extensiones que amplían su funcionalidad en función de las necesidades del desarrollador. A continuación se listan los módulos más importantes que utiliza ReNoLabs y la funcionalidad que aportan:

- Express: Es un marco de desarrollo de aplicaciones web mínimo y flexible que permite crear fácilmente sitios web.
- Passport y Passport-local: Añaden un middleware que se encarga de la autenticación de usuarios [12].
- Fs: Habilita el acceso al sistema de archivos del servidor.
- EJS (Embedded JavaScript): Es un lenguaje simple basado en plantillas que permite la generación de HTML utilizando JavaScript [13].
- Socket.io: Aporta webSockets full-duplex para la comunicación entre el servidor y el cliente.
- Child-process: Facilita las comunicaciones entre procesos (utilizado en controladores implementados sobre *Raspberry Pi*).

- ADS: Posibilita las comunicaciones con TwinCAT PLC (utilizado en laboratorios en donde se usa este software).

Además, en el servidor se encuentra una base de datos local simple que almacena los datos relativos a los usuarios que pueden acceder al laboratorio, implementada en un fichero de texto plano. Debido a la simplicidad del fichero, esta base de datos se gestiona directamente con Express aunque esta no es la única opción. Node.js dispone de multitud de extensiones que permiten las comunicaciones con diferentes tipos de bases de datos. Estas extensiones permiten conexiones con bases de datos tan dispares como bases de datos relacionales (por ejemplo SQLite o MySQL) o no relacionales (como LevelDB). [14]

### 2.4.2 Herramientas de Cliente

A pesar de que una vez implantado el laboratorio remoto en un servidor, la única herramienta necesaria por parte del cliente es un explorador de internet, el diseño de la interfaz gráfica debe ser creado por el diseñador del laboratorio. Esta interfaz deberá ser implementada, como ya se ha comentado, en HTML5 (para permitir el acceso al cliente con distintos dispositivos) y con una distribución sencilla y clara. Para el diseño e implementación de esta interfaz se ha escogido EjsS puesto que es una herramienta fácil de manejar y conocida entre los diseñadores de laboratorios de control [6, 7]. EjsS  (Easy Java/JavaScript Simulations) es una herramienta creada por Francisco Esquembre, profesor de la *Universidad de Murcia*. y denominada EJS, Ejs o EjsS [10]. Sin embargo, para evitar confusiones con EJS (Embedded JavaScript) que es utilizado en las herramientas del servidor, en la totalidad de este documento se denominará EjsS a la herramienta de cliente y EJS a la herramienta de servidor.

La UCM no es la única que ha sabido sacar provecho a EjsS como generador de interfaces dentro del marco de trabajo de ReNoLabs. La UNED, dentro de su portal UNILabs comentado anteriormente, también utiliza esta herramienta para la generación de los interfaces de los laboratorios remotos que pone a disposición de sus usuarios.

Como el próximo capítulo está dedicado íntegramente a esta herramienta, no se detallan más sus propiedades en éste.

# 3 EjsS: Ampliando Límites

EjsS es una herramienta gratuita y de código abierto, orientada a permitir a usuarios con pocos conocimientos de programación, la creación de simulaciones por ordenador, de un modo fácil, gráfico e intuitivo, destinada a estudiantes de ciencias, profesores o investigadores. EjsS está diseñada para que la persona que quiera crear una simulación pueda concentrarse la mayor parte del tiempo escribiendo y refinando los algoritmos del modelo científico objeto del estudio y el mínimo tiempo posible en técnicas de programación. EjsS, es una herramienta "generadora de código", cuyo resultado es el conjunto de elementos que conforman la propia simulación [10].

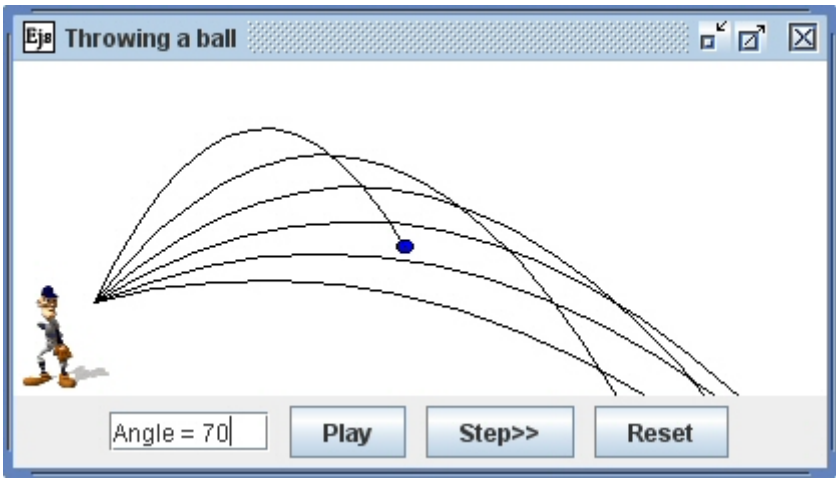


*Fig. 6. Ejemplo de simulación EjsS creada para Java [10]*

El código generado por EjsS puede estar bien en Java o JavaScript, lenguaje que se deberá seleccionar antes de crear la simulación y permitirá el uso de unos u otros elementos en función de la elección. Esta elección es muy importante ya que determinará el modo de distribuir la simulación y ser visualizada por el destinatario. En el caso de seleccionar JavaScript, el resultado podrá ser distribuido por internet y visualizado en cualquier explorador web estándar sin ningún tipo de requisito extra. En cambio, en el caso de crear la simulación en Java, su ejecución solo podrá verse en una máquina con el entorno de ejecución de Java (JRE).

Este trabajo utiliza EjsS para generar la interfaz gráfica de los laboratorios remotos en formato HTML por lo que se hará énfasis en describir las capacidades de EjsS para la generación de código JavaScript, aunque ofrece características similares en la generación para Java.

EjsS sigue el paradigma Modelo-Vista-Controlador que permite separar los datos y el comportamiento de la simulación de su representación visual. El desarrollo de la simulación separa el modelo de la simulación de la vista que lo representa en modo gráfico de modo que ambas se diseñan por separado. Para que el entorno gráfico sepa en todo momento cómo se debe mostrar el estado de la simulación, los elementos de la vista se enlazarán con las diferentes variables, funciones y elementos incluidos en el modelo. Durante la simulación, el motor interno de EjsS se encarga de mantener modelo y vista sincronizados.

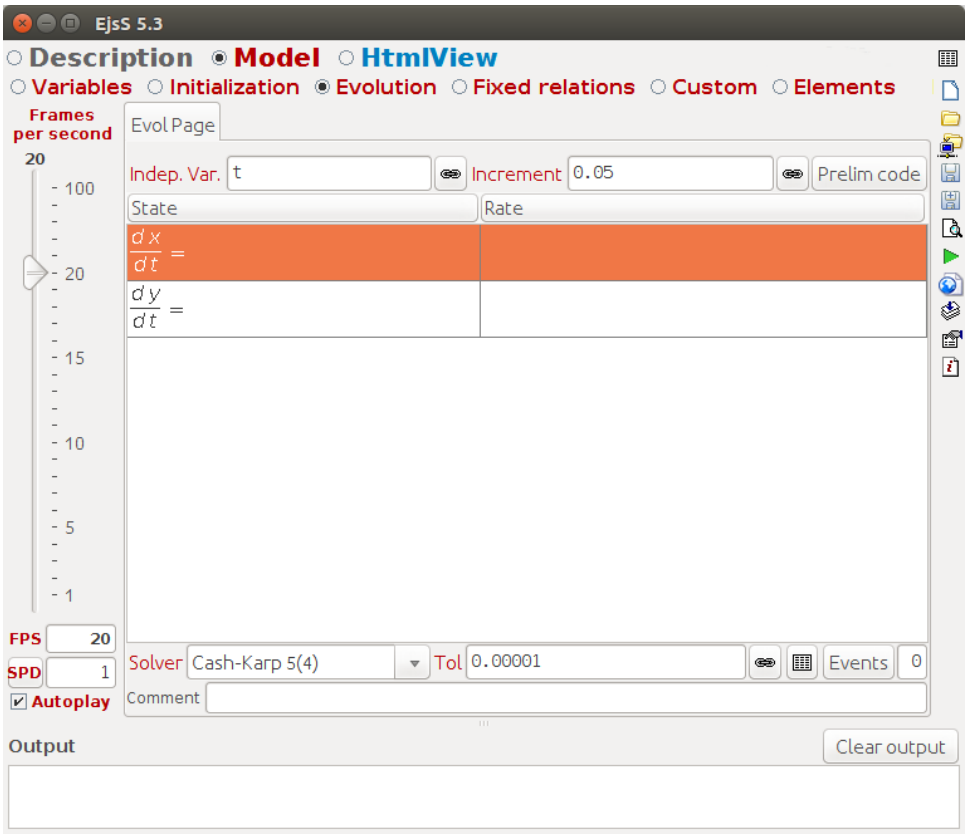


*Fig. 7. Página de evolución del modelo de EjsS*

Tal y como se muestra en la figura 7, EjsS muestra claramente esa separación en una distribución visual en forma de apartados que el desarrollador deberá rellenar.

- Descripción
- Modelo
- Vista HTML

La opción de descripción permite añadir a la simulación, páginas de texto enriquecido en las que el usuario podrá dar una descripción de la simulación que está creando. La creación de estas páginas no es obligatoria aunque sí muy recomendable para que el receptor de la simulación pueda entender mejor de qué se trata.

Bajo la opción de modelo se encuentran agrupadas también por apartados los diferentes tipos de elementos, véase la figura 7, con los que se irá construyendo el modelo científico de la simulación. Sin entrar en excesivo detalle, el modelo de la simulación está compuesto principalmente por un conjunto de variables sobre las que se pueden definir reglas de evolución bien mediante el uso de rutinas o bien mediante definiciones de ecuaciones diferenciales y un conjunto de relaciones fijas que se deben cumplir en todo momento.

La opción de vista HTML es donde se diseña y se implementa la visualización de la simulación. EjsS integra una cantidad considerable de elementos que pueden insertarse en la página. Estos elementos abarcan desde simples controles de usuario como cajas de texto o botones, hasta elementos más complejos de dibujo en 2D y 3D. La construcción de la página se hace mediante un control de tipo árbol que estructura los elementos de la página de un modo jerárquico y muy intuitivo, en la figura 8 puede verse un ejemplo. En este árbol se pueden ir arrastrando los

componentes seleccionándolos de los grupos de elementos de vista. Éstos elementos visuales serán los que se enlazarán con los elementos del modelo para que representen en todo momento el estado de la simulación durante la ejecución.

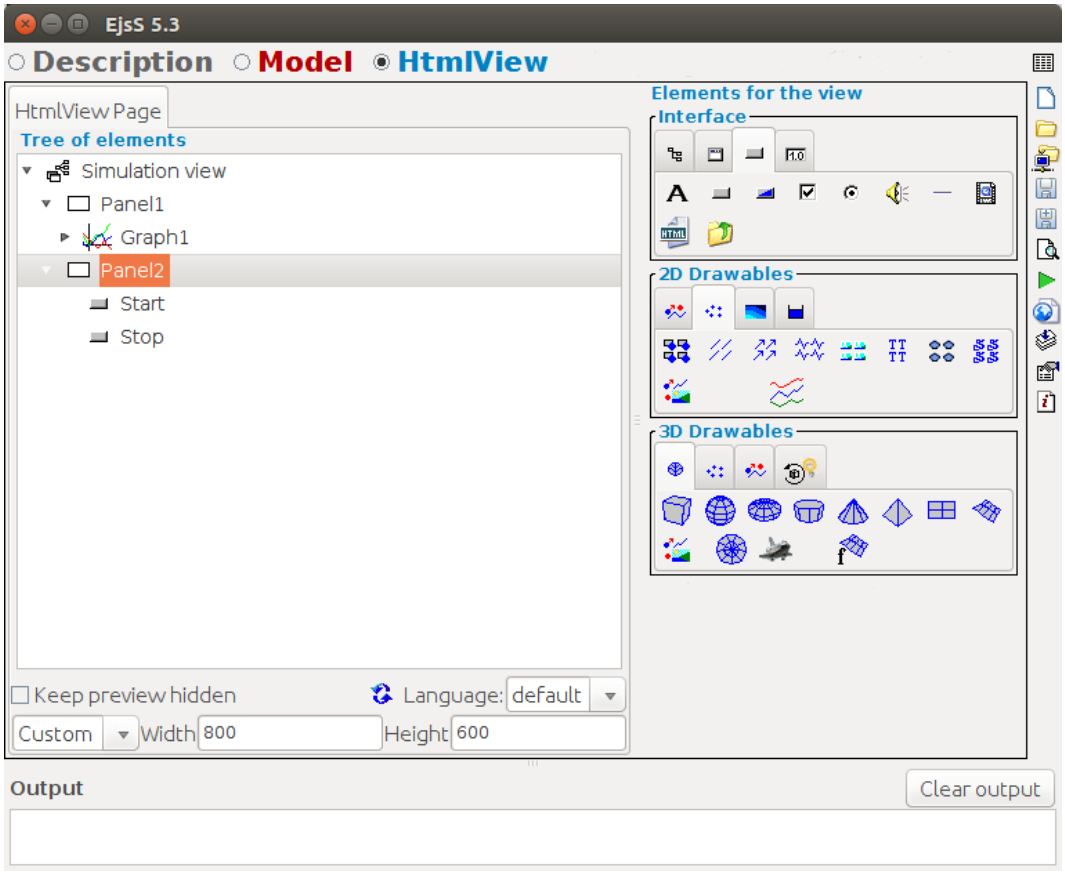


*Fig. 8. Pantalla de vista HTML de EjsS*

EjsS también ofrece al usuario una ayuda visual para que pueda ir viendo el resultado de la página que está creando en una previsualización mostrada en un explorador de internet que EjsS podrá ejecutar automáticamente. Esta previsualización carecerá de la funcionalidad interna de la simulación pero será suficiente para la creación de la vista.

## 3.1 Extensiones en EjsS

Una característica interesante acerca del modelo de EjsS es que permite a un usuario experimentado crear y añadir al entorno nuevos objetos de modelo que podrán ser utilizados para el modelado de la simulación. Estos elementos son extensiones que EjsS permite añadir a su entorno. Actualmente la herramienta está dotada de varias de estas extensiones para poder añadir al modelo elementos tales como librerías numéricas, acceso a algunos sensores si están presentes en el sistema o incluso acceso a software externo.

Las extensiones en EjsS se encuentran en carpetas específicas dentro de la estructura de carpetas de la instalación de la aplicación. Existe una carpeta para contener las extensiones para las simulaciones que se generan para Java y otra carpeta para contener las destinadas a las simulaciones que se generan para JavaScript. Una extensión está compuesta por una carpeta en donde se encuentra un archivo .jar que contiene las clases que implementan los elementos de modelo, y una serie de ficheros auxiliares como pueden ser los iconos para cada elemento o los ficheros de ayuda.

Al arrancar la aplicación, EjsS explora las carpetas que contienen las extensiones y añade al entorno dinámicamente y en tiempo de ejecución todos los elementos de modelo que se encuentren dentro del archivo .jar, que serán aquellas clases Java que se encuentran dentro del fichero que implementan una interfaz concreta. Éste es un mecanismo muy potente que permite a un usuario añadir o eliminar elementos de modelo de un modo muy sencillo en función de sus necesidades. Además, al ser EjsS una aplicación de código abierto, no resultará excesivamente complicado a un programador con experiencia en lenguajes de programación orientados a objetos implementar sus propios elementos de modelo.

Además de los elementos de modelo, EjsS dispone de un método muy sencillo que permite a un usuario crear elementos de vista personalizados y almacenarlos como elementos de usuario que pueden ser añadidos a otras páginas o proyectos EjsS. Para ello, el usuario solo debe copiar y pegar del árbol de elementos, el nodo que contiene la estructura que interesa guardar, dentro de la pestaña de elementos compuestos, que es la primera pestaña del grupo *Interface* y proporcionar un nombre y un icono que lo represente. Una vez hecho esto, EjsS toma la estructura completa de nodo y la almacena en formato XML en una carpeta concreta bajo su *workspace* y de ese modo se hace accesible a todas las simulaciones de ese tipo (Java / JavaScript) que se creen en el equipo. Estos nuevos controles se pueden añadir al árbol de vista igual que el resto de elementos, sin embargo, al insertar este elemento, automáticamente se añadirán los elementos de los que se compone en el mismo orden y con las mismas propiedades con las que fueron creados.

## 3.2 Limitaciones del modelo de extensiones de EjsS

Llegado el momento de abordar el trabajo de automatización de laboratorios remotos sobre ReNoLabs, se comenzaron a realizar diversas pruebas de concepto con desarrollos basados únicamente en las extensiones de elementos de modelo y vista que EjsS ofrecía. Lamentablemente, a pesar de que las funcionalidades implementadas cumplían las expectativas y se veía un grado de avance hacia el objetivo de la automatización perseguido, el resultado era un producto poco coherente con el resto de la herramienta y los elementos parecían estar en un lugar poco natural. Las extensiones de modelo estaban pensadas para añadir elementos que actúen en la simulación y puedan ser referenciados desde los elementos de vista, sin embargo, se estaba añadiendo funcionalidad con fines totalmente distintos, como la transmisión del propio código generado por EjsS a un servidor. Esta implementación, aunque funcional, rompía por completo la filosofía con la que fueron creadas las extensiones.

Además de posibilitar las comunicaciones por medio de elementos de modelo, se quería ofrecer al profesorado una serie de elementos visuales comunes en sus laboratorios. La aproximación inicial era obviamente crear elementos compuestos personalizados. Estos elementos tenían sentido únicamente en el ámbito de los laboratorios remotos por lo que debían ser instalados junto a las extensiones de modelo, sin embargo, estas extensiones de modelo y vista no están relacionadas en el ámbito de EjsS y la distribución e instalación de estas extensiones en nuevos equipo resultaba poco cómoda.

## 3.3 Nuevo rumbo

En este punto se mantuvo una reunión con Francisco Esquembre que puso de manifiesto el problema y en esa misma reunión nació la idea de dotar a EjsS con un nuevo modelo de extensiones llamados *Plugins* que permitirían añadir funcionalidades nuevas a EjsS de una forma más coherente e integrada sin necesidad de re-programar la aplicación. Los *Plugins* podrían ampliar o incluso

modificar el entorno EjsS sin necesidad de modificar el código original, permitiendo así explorar nuevas vías de integrar nuevas funcionalidades. Además, esta característica podría animar más a los usuarios de EjsS a crear sus propias extensiones ya que estas serían más resistentes al cambio de versión de EjsS tras una actualización ya que el único código que deberían modificar sería el que ellos mismos crearon y que conocen, sin necesidad de navegar por el código de EjsS para ver qué es lo que ha cambiado que hace que falle.

Esto suponía un reto ya que pensar a priori en todas las formas en las que un desarrollador puede necesitar o querer ampliar el sistema es completamente imposible. Aún así había que encontrar un método lo suficientemente flexible como para afrontar una primera aproximación de modo satisfactorio.

Esta reunión dio una perspectiva totalmente nueva al modo en el que se debía abordar la automatización de los laboratorios remotos, dando un giro inesperado a los desarrollos al tener que equilibrar el esfuerzo dedicado entre implementar la nueva capacidad en EjsS y añadir las capacidades que inicialmente estaban planificadas para ReNoLabs. Finalmente, este cambio de rumbo resultó en una reducción del número de nuevas funcionalidades implementadas en ReNoLabs pero en un gran avance a muestro modo de ver, en el trabajo de mejora de una herramienta ya de pos sí excelente como es EjsS.

## 3.4 Plugins

La nueva capacidad de *Plugins* de EjsS tiene como objetivo permitir la ampliación de funcionalidad que ofrece la propia herramienta así como la adaptación del entorno para adecuarlo a un "escenario" o "problema" específico que cualquier usuario pueda tener.

Además de cumplir estos objetivos, el mecanismo de extensión de EjsS que se ha propuesto en la realización de este trabajo se ha desarrollado tratando de satisfacer los siguientes requisitos personales:

- Sencillo de distribuir: Habría que poder hacer llegar el *Plugin* a los destinatarios que vayan a requerir la funcionalidad de la manera más sencilla posible y sin ningún tipo de complicación.
- Autocontenido: Un *Plugin* debe contener todo lo que se espera de él sin dependencias de contenido externo.
- Rápido de desinstalar: Es importante que de forma sencilla se pueda eliminar cualquier traza relacionada con un *Plugin* en el caso de que fallase o tuviese cualquier tipo de problema de compatibilidad bien con la versión de EjsS (debido a una actualización) o bien con cualquier otra extensión que pudiera estar instalada.

Además de estas características, en todo momento se ha tenido en mente el tratar de realizar todo el desarrollo siendo lo menos intrusivo posible con el código fuente de EjsS y tratando de seguir la filosofía de desarrollo utilizada hasta ahora por la aplicación.

En este trabajo se han considerado de interés cuatro puntos de extensión:

- Barra de botones
- Opciones principales
- Modelo
- Grupos de controles HTML

Los siguientes apartados describen en detalle en qué modo permiten estos puntos extender la funcionalidad de EjsS.

En el caso de querer conocer más en detalle la instalación y gestión de los Plugins así como los elementos necesarios para su desarrollo se puede consultar el Anexo A. Además, para conocer algunos aspectos importantes con respecto a la versión de EjsS en el que se pueden utilizar, se recomienda consular el Anexo B.

### 3.4.1 Barra de botones

La barra de botones de EjsS agrupa una serie de funciones genéricas que permiten gestionar la simulación y el propio entorno pero sin estar ligadas al contenido de la propia simulación. En esta barra se encuentran los botones que permiten por ejemplo, crear, cargar, modificar y guardar simulaciones, generar el código de las simulaciones y configurar los diferentes parámetros de generación. La figura 9 muestra los botones que ofrece EjsS.

*Fig. 9. Barra de botones de EjsS*

Los botones de esta barra pueden ser de dos tipos:

- Aquellos que realizan la acción programada en el momento en el que son pulsados como el botón de crear nueva simulación o salvar la simulación. Esta acción se dispara pulsando el botón izquierdo del ratón sobre el botón elegido de la barra.
- Aquellos que despliegan un menú contextual que agrupa un número de acciones que el usuario puede seleccionar. La lista de acciones se despliega pulsando el botón derecho del ratón. Habitualmente, estos botones tienen programada una acción por defecto asociada al botón izquierdo que se ejecutará inmediatamente sin mostrar el menú contextual. En este grupo encontramos, por ejemplo, el botón de generación de la simulación que despliega diferentes opciones o el botón de cargar una simulación que muestra una lista con las últimas simulaciones utilizadas y nos permite cargar rápidamente una de ellas sin necesidad de navegar por el sistema de carpetas.

Los *Plugins* permiten añadir en esta barra botones de ambos tipos para dotar a EjsS de la funcionalidad que resulte apropiada para la extensión que se desea realizar o incluso para suplir alguna carencia que se considere que la herramienta pudiera tener. Las posibilidades son amplias.

Por ejemplo, supongamos que un usuario desea poder enviar una simulación por correo electrónico sin salir del entorno y sin necesidad de tener que preocuparse por encontrar en la estructura de carpetas el fichero que contiene la simulación. Salvando las dificultades de integración con software de terceros, un *Plugin* permitiría añadir un botón a la barra que abriera algún programa externo de correo electrónico y adjuntara automáticamente el archivo de simulación al correo.

O supongamos que una universidad cuenta entre las prácticas obligatorias de alguna asignatura, la creación de alguna simulación en EjsS. Para facilitar las entregas de estas prácticas, la universidad pone un servidor para que el alumno pueda realizar la entrega en remoto. Esta entrega también sería susceptible de ser automatizada e integrada en un nuevo botón en EjsS por lo que la universidad podría distribuir este *Plugin* entre sus alumnos.

En el próximo capítulo mostraremos un ejemplo de utilización de estos botones para el despliegue automático de nuestro laboratorio.

### 3.4.2 Opciones principales

La barra de opciones principales define una serie de conjuntos de información de los que se compone una simulación de EjsS y que son propios de la simulación. A diferencia de la barra de botones que está orientada a gestionar el entorno general de EjsS, la información que se muestra en estas opciones se almacena en el archivo de la simulación.

Estas opciones encierran el paradigma modelo-vista-controlador en el que se basa la totalidad de la herramienta. Desde ese punto de vista, resulta difícil pensar en elementos adicionales que pudieran añadirse a este conjunto, sin embargo, sí se pensó que utilizar únicamente la barra de botones para ampliar la funcionalidad podría resultar limitante para un desarrollador creativo, por lo que se ha optado por permitir añadir libremente nuevas opciones a las existentes. Evidentemente, a pesar de que este conjunto de opciones principales existe para definir propiedades de la simulación, la implementación y objetivo de cada una de estas nuevas opciones es responsabilidad exclusiva de su desarrollador, por ello, el mecanismo de extensiones no hará ningún tipo de comprobación de su contenido, dando completa libertad al desarrollador en este aspecto.

En este aspecto también las posibilidades son amplias. Imaginemos que la solución a implementar sobre EjsS requiere determinados parámetros de configuración específicos para cada simulación. Para resolver este problema se podría implementar un *Plugin* que añada una nueva opción a estas opciones en donde el usuario pueda configurar los nuevos parámetros de simulación y que se almacenaran en el archivo de la simulación junto a todos los demás.

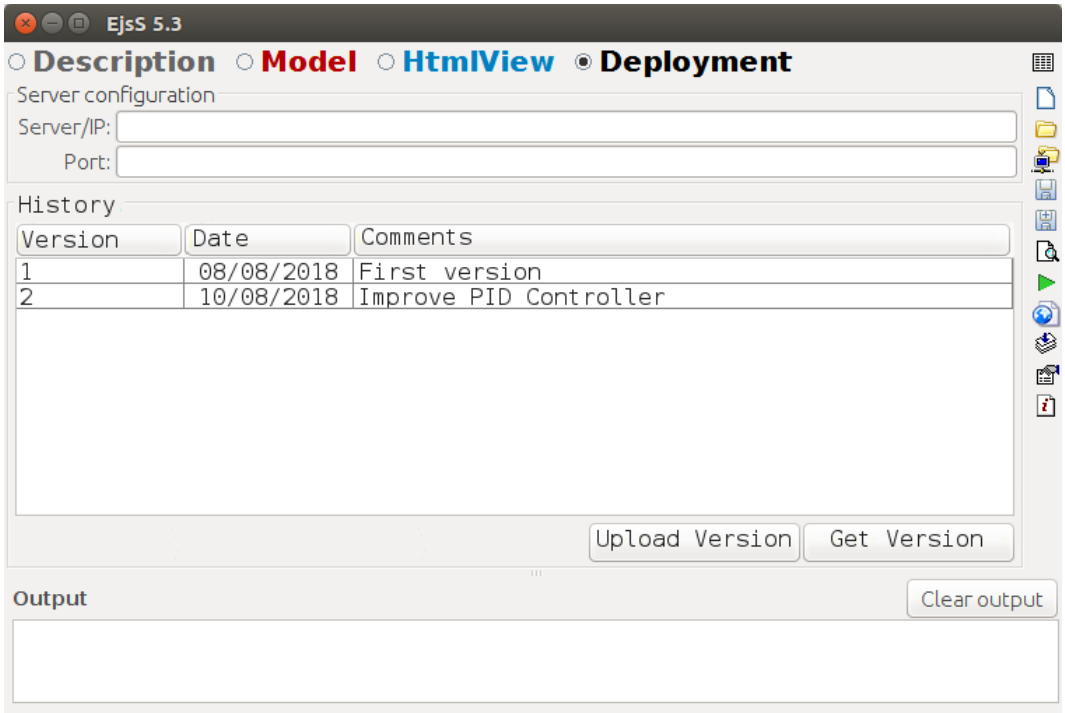


*Fig. 10. Ejemplo de extensión de Opciones Principales*

Volviendo al ejemplo de las entrega de las prácticas de EjsS del apartado anterior, supongamos que la universidad ofrece a cada alumno la posibilidad de ir guardando su progreso en una carpeta individual. Al tratarse de una funcionalidad desligada de la simulación, la primera aproximación podría ser la de añadir un par de botones para estas acciones. Sin embargo, imaginemos que para dar valor añadido, cada vez que el alumno envíe una actualización se le dará la opción de añadir un comentario que se almacenará en un histórico que el alumno podrá consultar con posterioridad.

Además, cada versión que se envía se almacena en el servidor de modo independiente con la intención de permitir al alumno recuperar cualquiera de estas versiones. Esto podría lograre mediante una segunda opción, basada en el uso de un Plugin de opciones principales similar al que se representa en la figura 10.

A pesar de que todo esto podría realizarse añadiendo botones a la barra, todo este proceso puede resultar más cómodo si el *Plugin* añadiera una nueva opción con los elementos gráficos que faciliten la tarea y puedan estar disponibles en todo momento.

### 3.4.3 Modelo

El modelo representa el corazón de la simulación en EjsS. Define los elementos que componen la simulación y su comportamiento. EjsS ofrece la posibilidad de programar y añadir nuevos tipos de elementos del modo descrito anteriormente. Los nuevos tipos de elementos se encuentran disponibles en un apartado dedicado dentro del modelo. Este mecanismo es suficiente en los casos en los que éstos sean los únicos elementos que conforman una nueva funcionalidad pero si estos elementos tienen que ir en consonancia con otras extensiones del modelo, vista u opciones principales, la implantación y mantenimiento de una nueva solución sobre EjsS se complica.

Además de esta dificultad, este modelo de extensión presenta otro inconveniente y es que estos elementos, tal cuál están concebidos, no pueden estar contenidos en un *Plugin*. Un *Plugin* podría perfectamente implementar la clase que extiende el modelo pero para que éste estuviera disponible en EjsS, tanto el archivo del Plugin desarrollado como todos los archivos auxiliares necesarios deberían instalarse dentro de la carpeta dedicada a las extensiones de modelo. Esta solución no estaría autocontenida en un único archivo dificultando la distribución de la solución lo que rompería con al menos dos de los requisitos personales propuestos para los *Plugins*.

Debido a estas razones, se ha diseñado la infraestructura de *Plugins* para permitir a éstos añadir nuevos apartados a los ya ofrecidos por el modelo. El permitir extender los apartados del modelo ofrece una ventaja añadida y es que al tener un apartado específico para cada tipo de elemento, se le puede dar una interfaz gráfica personalizada y más adecuada, visible en todo momento, haciendo que los nuevos elementos sean más cómodos de utilizar. Además, esta extensión permite implementar sobre EjsS soluciones más completas a la vez que cumple con todos los requisitos propuestos.

Esta extensión está pensada para añadir elementos de modelo que puedan ser referenciados por la vista y que se almacenen en el archivo de simulación, pero EjsS no forzará su contenido por lo que un desarrollador podría implementar otro tipo de funcionalidades más genéricas, a pesar de que sería más adecuado hacer esto dentro de las extensiones a las opciones principales.

Las extensiones que se pueden realizar son innumerables aunque hay que reconocer que no es debido a los *Plugins* sino a la funcionalidad original de EjsS. Lo que aporta esta extensión es un nuevo modo de agruparlas y distribuirlas dentro de un contexto de funcionalidad mayor. EjsS ofrece de serie un número de elementos entre los que se encuentran librerías numéricas y elementos que interactúan con otros componentes externos.

Retomando nuestro ejemplo de la práctica de universidad y entrando en detalle de la propia práctica, supongamos que el alumno debe realizar una simulación de una planta en la que hay que controlar los niveles de varios tanques de agua. El objetivo de la práctica consistirá en que el alumno elija los controladores que considere óptimos y que simule su comportamiento. Para esta simulación el alumno no solo tendrá que simular el comportamiento de los controladores sino que tendrá que recrear el comportamiento y estado interno de los tanques de agua. En este caso, al

profesor no le interesa que el alumno dedique tiempo al detalle del tanque así que para permitir que el alumno dedique el mayor tiempo y esfuerzo en la simulación del controlador, puede ofrecer en el *Plugin* un elemento de modelo que simule el comportamiento del tanque. Este componente podría tener como propiedades, la capacidad del mismo en litros y la altura total del tanque, como entradas el caudal entrante y el caudal saliente, y como salida podría indicar el nivel del líquido. Este tanque podría incluso tener entradas adicionales para introducir interferencias como obstrucciones en los conductos de entrada y de salida. El ejemplo podría complicarse todo lo deseado. El profesor podría crear todos los elementos que quisiera para conseguir que el alumno pueda centrarse en lo que es verdaderamente importante en la práctica dejando de un lado detalles que, aunque importantes, desvían la atención del alumno. La figura 11 muestra una implementación de esta extensión.

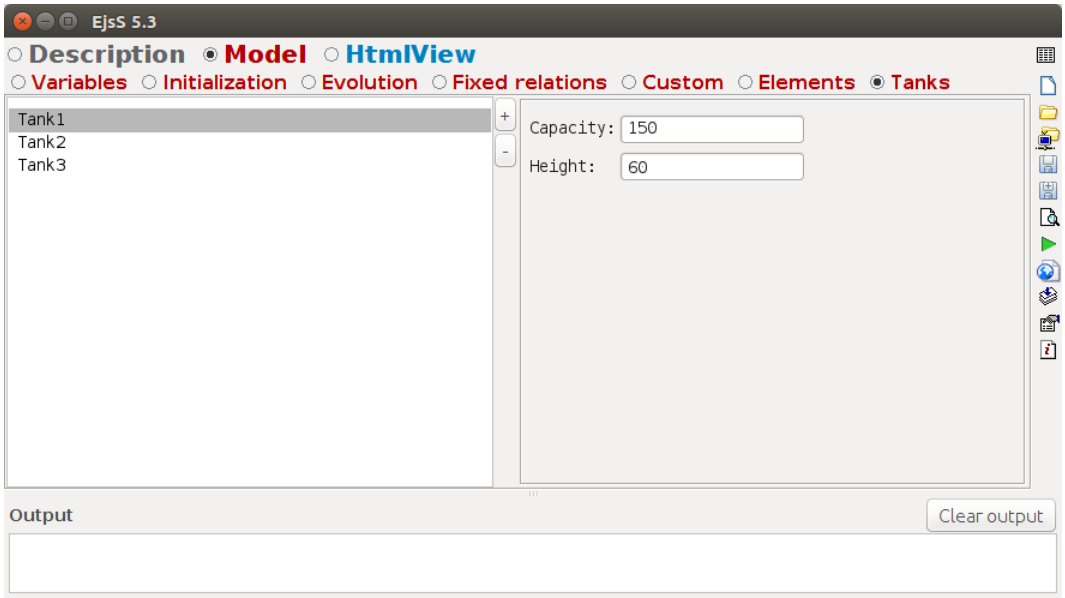


*Fig. 11. Ejemplo de extensión de Modelo*

Es importante tener en cuenta que en el ejemplo expuesto, el objetivo del profesor será facilitar la práctica al alumno, sin embargo crear un elemento de estas características requiere que el profesor tenga un buen conocimiento del lenguaje de programación (Java / JavaScript) en el que vaya a correr la simulación ya que puede no resultarle tan sencillo de programar.

### 3.4.4 Grupos de Controles HTML

Al igual que en el caso anterior, EjsS también dispone de un método para añadir controles HTML definidos por el usuario. Ésta es una herramienta mu útil que aporta EjsS, sin embargo, al querer que estos controles personalizados formen parte de una determinada funcionalidad extendida nos encontramos con los siguientes problemas:

- Desde el punto de vista de distribución, al estar ubicados los controles en un directorio independiente fijado en EjsS, resulta más compleja la implantación de una solución personalizada que quiera aportar sus propios elementos visuales. Del mismo modo, eliminar por completo todos los elementos aportados por la solución requerirá de varios pasos, no siendo compatible con los objetivos de simplicidad que persiguen los *Plugins*. Éste es un problema común con las extensiones de los elementos de modelo.
- Además del problema anterior, los elementos visuales personalizados tienen un inconveniente añadido que no tenían los elementos de modelo. Cuando el usuario añade el

elemento visual personalizado al árbol, EjsS añadirá todos los elementos de la estructura tal y como fueron guardados durante la creación, y todos sus elementos y propiedades quedarán expuestos. De no tener especial cuidado, tanto la estructura del elemento visual como cualquiera de sus propiedades puede ser modificadas haciendo que el comportamiento del conjunto no sea el esperado, pudiendo no ser trivial en todos los casos su corrección. Además, en el caso de que el componente pueda ser parametrizado, no siempre resultará sencillo adaptarlo a un nuevo ejercicio o simulación, sobre todo si el control está compuesto de muchos elementos a parametrizar.

Debido a estos problemas, se ha optado por añadir a la infraestructura de *Plugins* la posibilidad de añadir al entorno EjsS nuevos controles HTML. Esto tiene la ventaja de que al ofrecer la capacidad de añadir nuevos elementos visuales totalmente personalizados, todo el comportamiento esperado del control queda encapsulado en un único elemento visual que expone únicamente aquellas propiedades que se permiten parametrizar, delegando al propio componente la actualización de las propiedades de los elementos internos para su correcto funcionamiento. Además, al estar expuesto únicamente el componente principal, se elimina el riesgo de eliminar accidentalmente alguno de los elementos de los que está compuesto reduciendo la posibilidad de introducir algún error que impida el correcto funcionamiento del componente.

Evidentemente, no todo son ventajas y el principal inconveniente que tiene esta extensión es que hay que tener en cuenta que esta facilidad de encapsular toda la funcionalidad en un único elemento resta la capacidad al usuario de manejar el aspecto visual de cada elemento individual de los que se compone el control. Aunque siempre se puede utilizar los controles HTML como previamente se definen en EjsS en lugar de empaquetarlos en *Plugins*. Está claro que un componente puede exponer todas las propiedades que desee, pero sobrecargar al usuario con multitud de parámetros podría llegar a resultar contraproducente, por lo que en cada componente se debe llegar a un compromiso entre facilidad de uso y personalización.

Los elementos visuales que pueden añadirse pueden ser de lo más variados. Continuando el ejemplo de la práctica del tanque de agua mostrado en los apartados anteriores, en este caso el profesor podría querer aportar una implementación propia al elemento tanque del modelo. Los elementos gráficos complejos en EjsS se componen de muchos elementos y el profesor tampoco quiere que los alumnos se tengan que encargar de estos detalles así que tiene la opción de incluir un elemento gráfico en el *Plugin* que el alumno pueda utilizar en su simulación.

La figura 12 representa el modo en el que EjsS muestra las extensiones a sus elementos visuales. La representación gráfica de estos elementos dependerá de la implementación realizada en la librería JavaScript.

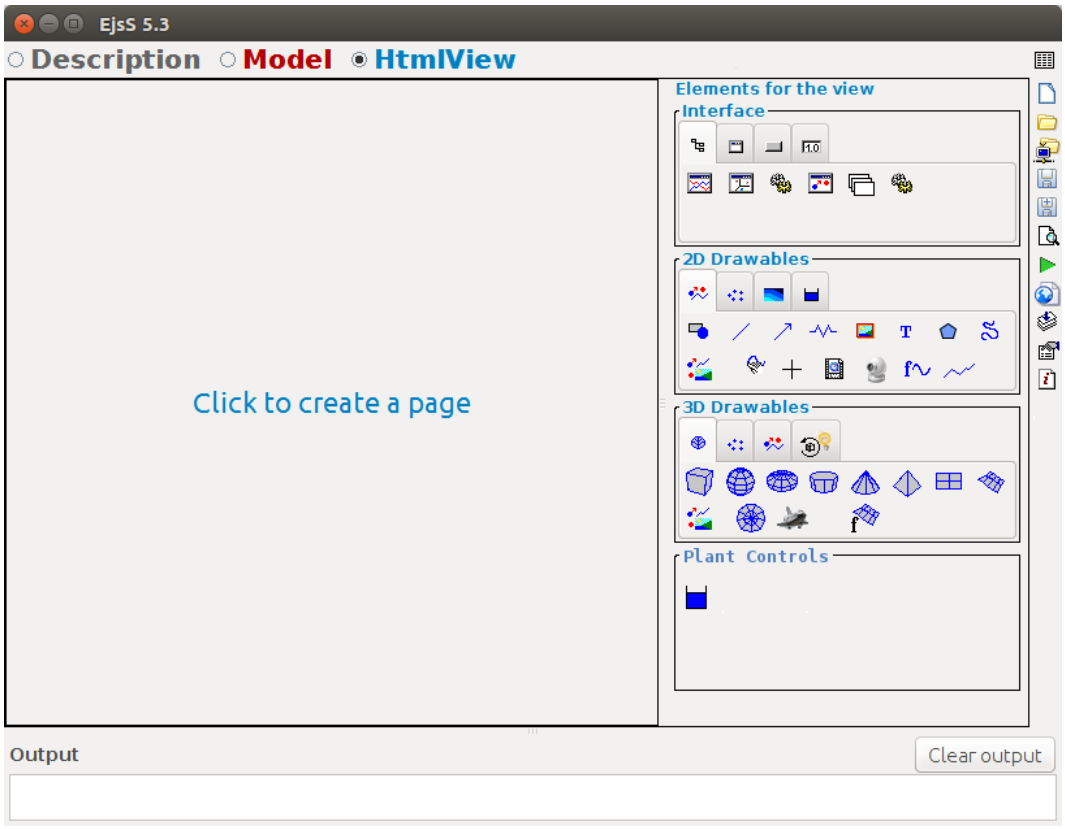


*Fig. 12. Ejemplo de extensión de Elemento Visual*

# 4 Solución Integrada para Laboratorios Remotos

La infraestructura de *Plugins* desarrollada y explicada en el capítulo anterior es la que se ha utilizado dentro del marco de ReNoLabs para implementar los desarrollos de funcionalidades destinadas a crear y gestionar los laboratorios remotos de la UCM. En concreto, estos desarrollos se han enfocado fundamentalmente en:

- Facilitar la creación de interfaces.
- Automatizar los trabajos de despliegue y mantenimiento de laboratorios.
- Aportar mayor control sobre el código desplegado en el laboratorio.

Con esta nueva infraestructura se ha creado un *Plugin* que extiende todos y cada uno de los puntos de extensión implementados para aportar una solución completa, coherente e integrada al problema de implementación y despliegue de laboratorios remotos. No hay que olvidar que a pesar de que son muchas las funcionalidades y mejoras que se pueden aportar al sistema, únicamente se ha implementado un conjunto básico de funcionalidades, con el objetivo de aportar una buena visión del potencial del uso de estas nuevas extensiones EjsS. Este conjunto de funcionalidades constituyen una primera aproximación que conformarán la base del desarrollo de una futura solución más completa.

Un laboratorio remoto en el marco ReNoLabs está compuesto por un conjunto de elementos software y equipos físicos. Recordando el esquema de laboratorios remotos implementado en la UCM mostrado en la figura 4, entre los elementos software se encuentran:

- Interfaz de usuario creada con EjsS.
- Aplicación Node.js.
- Controlador (C, TwinCAT, LabVIEW).

En cuanto a los equipos físicos, éstos serán diferentes en cada laboratorio y se enlazarán con el controlador en función de la plataforma que se haya elegido para su implementación.

En este trabajo se ha tratado de aportar beneficios a todos los tipos de laboratorio, no obstante, en lo que al controlador de laboratorio se refiere, debido a las grandes diferencias que puede haber en los diferentes laboratorios, este trabajo está centrado concretamente en aportar mejoras al software de los laboratorios remotos implementados con un controlador en C, aunque lo descrito a continuación podría extenderse fácilmente a otros entornos. El *Plugin* creado en la parte cliente añade mejoras sustanciales aunque también se ha tenido que realizar alguna modificación en la aplicación Node.js para soportar las nuevas funcionalidades implementadas en el cliente.

Los siguientes apartados describen en detalle las funcionalidades implementadas, indicando el modo en el que contribuyen a la solución final y, respecto a los desarrollos de mejora realizados en EjsS, cuales son los puntos de extensión utilizados.

## 4.1 Modelo del laboratorio

En ReNoLabs, un laboratorio remoto está formado por un conjunto fijo de propiedades como las señales de entrada y las señales de salida. Estas señales se intercambian entre el controlador del laboratorio y la interfaz gráfica a través de la librería de comunicaciones *socket.io* desde la

aplicación Node.js y son las que describen en todo momento el estado interno del laboratorio remoto.

El corazón de una simulación en EjsS es sin duda alguna el modelo. En él se definen la totalidad de variables y elementos que forman parte de la simulación y su comportamiento. En el caso de un laboratorio remoto, lo esperado sería que existiera algún tipo de elemento o conjunto de elementos que lo representaran. Sin embargo, en ReNoLabs la gestión de las señales que representan el estado interno del laboratorio remoto se hace de un modo poco claro a través de definiciones manuales de objetos y procedimientos JavaScript en los distintos apartados que EjsS ofrece para la definición del modelo. Además, parte de la gestión de estas señales se realiza mediante llamadas explícitas a la librería *socket.io* que no son reconocidas por EjsS y que provocan que falle la generación de código. La solución de ReNoLabs a este problema es mantener estas líneas comentadas para posteriormente ser descomentadas tras la generación. Este proceso manual no resulta en absoluto ideal.

Para dar solución a los problemas tanto de uso de señales como de procesos manuales tras la generación, se ha implementado dentro del *Plugin* una extensión de modelo que permitirá crear en EjsS, objetos que representen a los laboratorios remotos. Estos nuevos objetos agrupan todas las propiedades de un laboratorio remoto en un objeto de modelo que podrá ser utilizado para enlazar los elementos visuales. La figura 13 muestra esta extensión.

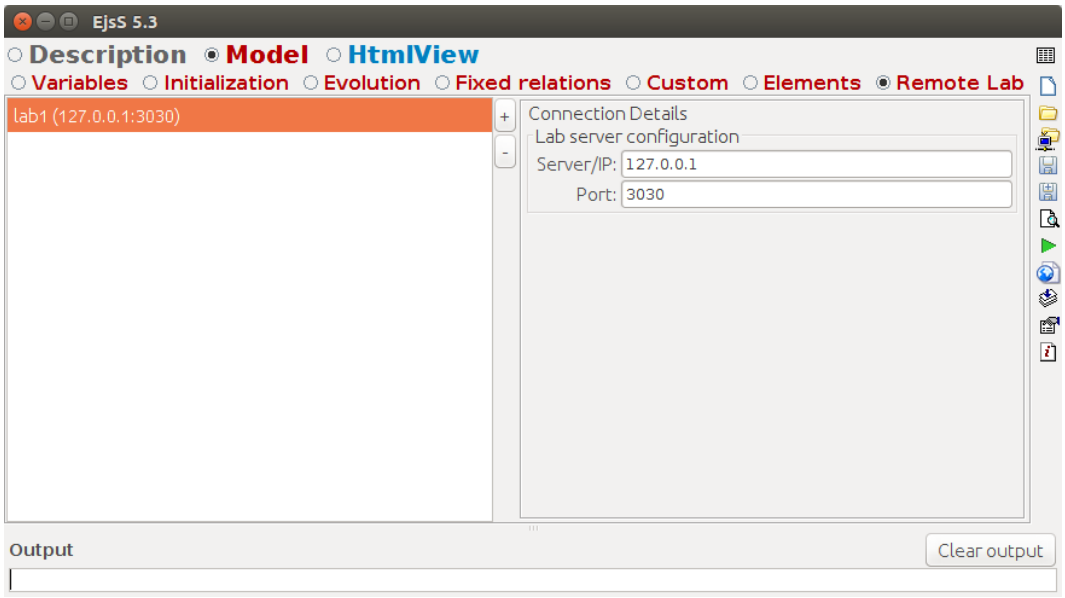


*Fig. 13. Extensión Remote Lab*

El nuevo apartado aparece bajo el nombre *"Remote Lab"* en la sección del modelo. En esta pantalla se añaden y configuran los elementos de laboratorio. Normalmente la interfaz gráfica se genera para un único laboratorio por lo que sería suficiente con un único objeto de este tipo. Sin embargo, no se ha restringido la cantidad de laboratorios que el usuario puede añadir. Esto permite crear interfaces gráficos "conectados" a más de un laboratorio haciendo posible, por ejemplo, la creación de una interfaz de monitorización de un conjunto de laboratorios.

La configuración de estos elementos es muy simple, cada elemento únicamente necesita conocer la dirección del servidor del laboratorio y el puerto en el que se conectará. A partir de ahí, el propio objeto se encargará en tiempo de ejecución de las comunicaciones con la aplicación Node.js remota.

La pantalla de esta extensión de ReNoLabs, mostrada en la figura 13, está dividida en dos partes. En la parte izquierda se muestran una lista con los elementos de laboratorio remoto añadidos y en la

parte derecha se muestran las propiedades del elemento seleccionado de la lista. Una vez rellenados los valores de la conexión, éstos datos también aparecerán en la lista junto al nombre asignado para que el usuario pueda consultar con más facilidad y de un solo vistazo, los laboratorios que tiene configurados sin necesidad de ir seleccionándolos uno a uno.

Los elementos de laboratorio remoto tienen sus propias propiedades y métodos que permiten por un lado conocer el estado del laboratorio remoto y por otro manipularlo realizando funciones como arranque, reseteo o parada. Las propiedades de estos elementos son:

- *address*: almacena la dirección del servidor en donde se encuentra el laboratorio remoto.
- *port*: almacena el puerto de conexión.
- *state_EJS*: contiene los valores de las señales de entrada al laboratorio remoto que se desea enviar desde EjsS.
- *state_REAL*: contiene los valores del estado completo del laboratorio remoto. Esta propiedad contiene a su vez las siguientes propiedades:
  - config: almacena el estado de ejecución del laboratorio.
  - evolution: contiene los valores de las señales de salida del laboratorio
  - Adicionalmente, este objeto contiene los valores de las señales de entrada aceptadas por el laboratorio y que están en uso en cada momento.

Por otro lado, los métodos de los que se componen estos elementos son:

- *send_connect*: esta función se encarga de gestionar el estado de ejecución del laboratorio remoto. Acepta un parámetro de entrada que será la función que se desea realizar. Los valores posibles son: 0 para parar el laboratorio, 1 para arrancarlo, 2 para arrancar la ejecución, 3 para pausar la ejecución y 4 para reiniciar el estado del laboratorio remoto.
- *update*: esta función se encarga de enviar al laboratorio remoto los valores de las señales de entrada que se desean actualizar.

Para enlazar estos elementos a los elementos visuales dentro de EjsS normalmente se utilizarán únicamente las propiedades, ya que los métodos los utilizarán otros componentes implementados dentro del *Plugin*. Si se consulta el código generado se verá que estos elementos poseen más propiedades y métodos, sin embargo, estos deberían considerarse privados, por lo que no se detallan en este documento.

## 4.2 Interfaz del Laboratorio

La interfaz del laboratorio remoto es la que le será presentada al usuario del mismo y que constituirá el único punto desde el que se podrá interactuar remotamente con el laboratorio tanto para visualizar su estado como para gestionarlo. Esta interfaz está compuesta por una página web que se podrá visualizar en cualquier explorador web estándar.

EjsS dispone de una herramienta muy potente para diseñar y establecer los elementos gráficos que se mostrarán en una página HTML. ReNoLabs ha sabido tomar partido de ella y será la herramienta que se utilizará para crear la interfaz de usuario de los laboratorios remoto.

El conjunto de elementos HTML de los que originalmente dispone EjsS es capaz de cubrir las necesidades de los laboratorios remotos creados en la UCM. Sin embargo, al tratarse de elementos genéricos, tienen la desventaja de tener que ser personalizados cada vez para los nuevos

laboratorios. Las páginas HTML de los laboratorios creados por la UCM están compuestas por muchos elementos, y en muchos casos, se componen de conjuntos de controles a los que se les ha dado un aspecto visual y un comportamiento que no difiere mucho entre un laboratorio y otro. El hecho de tener que posicionar y personalizar todos esos elementos para cada laboratorio suponía un esfuerzo repetitivo y poco gratificante por lo que se estudió la posibilidad de automatizar de alguna manera la creación de estos elementos. Durante la realización de este trabajo se han identificado dos elementos que pueden ser considerados dentro de este tipo y se han implementado como una extensión de elementos visuales como parte del *Plugin*. Estos elementos agrupan todos los componentes individuales y se gestionan dentro de EjsS como si de un único elemento se tratara pudiendo acceder fácilmente a sus propiedades para establecer el aspecto visual del conjunto desde un único lugar.

Estos elementos son:

- El elemento de control de laboratorio.
- El elemento de selección de función.

Además de estos dos elementos visuales, se ha añadido un tercer elemento visual, inexistente hasta ahora, pero que juega un papel muy importante en el proceso de automatización tal y como se describirá posteriormente. Este elemento será el encargado de gestionar la conexión y autenticación con el laboratorio remoto cuando éste aún no haya sido desplegado en el servidor. Este nuevo control también facilitará las pruebas y el mantenimiento de un laboratorio remoto existente, dando mayor grado de garantía de funcionamiento de las modificaciones en una fase previa a su despliegue y sin necesidad de dejar fuera de servicio el laboratorio original. La figura 14 muestra estos nuevos elementos visuales añadidos en EjsS.

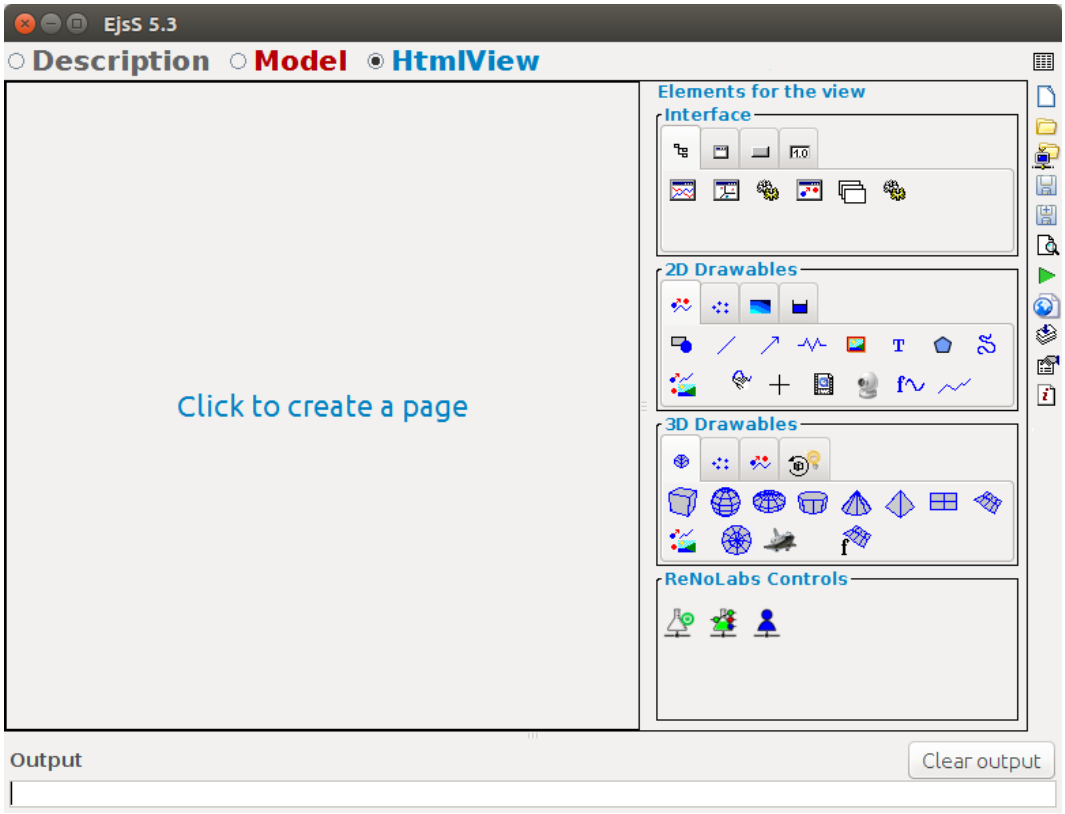


*Fig. 14. Extensión Elementos Visuales ReNoLabs*

Con la creación de los nuevos controles HTML se ha logrado no solo facilitar la creación de diversos elementos, sino reducir considerablemente el tiempo requerido para la creación de nuevos laboratorios, sin mencionar la fiabilidad que aporta el saber que el grupo de controles se va a comportar de igual modo en todos los laboratorios en los que estén desplegados al haber eliminado el riesgo de haber omitido algún parámetro de configuración en alguno de sus componentes.

Otra ventaja que ofrece la extensión de elementos visuales es que para modificar el comportamiento de un control de laboratorio, únicamente habría que modificar la librería JavaScript asociada sin necesidad de volver a desplegar el laboratorio completo.

Al igual que se han creado estos tres elementos, podrían haberse implementado muchos más, pero el conjunto de elementos seleccionado sirve para demostrar la validez de esta solución. A continuación se describen estos elementos con más detalle.

### 4.2.1 Login

El nuevo elemento creado encargado de la autenticación de usuarios en los laboratorios remotos es el llamado *LabLogin*. Este control es relativamente sencillo en cuanto a apariencia pero realiza una labor muy importante durante la fase de desarrollo de una interfaz de laboratorio. Este control permitirá al desarrollador introducir el nombre de usuario y su contraseña en tiempo de ejecución y el propio control se encargará de conectarse al laboratorio o laboratorios a los que esté asociado para realizar la autenticación en el sistema.

Una vez añadido este elemento a la vista HTML su aspecto visual será similar al mostrado en la figura 15.


*Fig. 15. Aspecto Visual de LabLogin*

Para utilizar el control, se introducirán los datos de autenticación que son el nombre de usuario y la contraseña y se pulsará el botón situado a la derecha tras la caja de texto de la contraseña. En los casos en los que la autenticación haya sido exitosa, se crearán conexiones con todos los laboratorios en los que el control haya logrado autenticarse. Estas conexiones serán utilizadas posteriormente por los demás elementos enlazados con esos laboratorios. En el caso en el que se haya autenticado exitosamente en algún laboratorio, los elementos de nombre de usuario y contraseña se deshabilitarán y el icono del botón pasará a ser de color rojo. Es importante tener en cuenta que no se generará ninguna conexión para ningún laboratorio en el que no se haya autenticado bien porque se haya rechazado la conexión o porque el laboratorio no haya respondido.

Para realizar la desconexión de los laboratorios se pulsará de nuevo el botón. Una vez desconectado de los laboratorios, las cajas de texto del nombre de usuario y de la contraseña se habilitarán de nuevo para quedar preparados para un nuevo uso y el icono del botón volverá a ser de color verde.

Para añadir este elemento a la interfaz, bastará con seleccionar el elemento dentro del grupo de controles visuales de ReNoLabs dentro de la vista HTML y arrastrarlo al árbol, dentro de cualquier elemento de tipo contenedor y asignarle un nombre único. Una vez allí podrán configurarse sus propiedades.

Las propiedades que este elemento expone y que se presentan en la figura 16 son:

- *Labs*: establece la lista de laboratorios en los que realizará la autenticación. Estos laboratorios deberán ser elementos contenidos en la lista de laboratorios de la extensión de modelo.

- *Font*: establece el tipo y tamaño de fuente que utilizarán las etiquetas del elemento.


*Fig. 16. Propiedades de LabLogin*

Este control se puede asociar a más de un laboratorio. Esta característica ofrece la posibilidad de crear páginas en la que se puedan manejar y visualizar información relativa a varios laboratorios a la vez, minimizando el uso de controles ya que se lograría con un único elemento *LabLogin*.

Anteriormente se ha comentado que este control es de gran utilidad durante la fase del desarrollo debido a que permite la autenticación en el sistema que aloja el laboratorio remoto. No obstante, hay que destacar que cuando la interfaz se encuentra alojada en el servidor, la autenticación en el sistema estará controlada por la aplicación Node.js y la correcta autenticación del usuario estará garantizada a la hora de visualizar la página ya que el alumno ha debido pasar obligatoriamente por la página de autenticación. La figura 17 muestra un ejemplo de página de laboratorio servida por la aplicación Node.js tras la autenticación. Como se puede apreciar, el usuario conectado se muestra en la esquina superior derecha. Esto hace que *LabLogin* sea totalmente innecesario en el servidor. En previsión a esta situación y para evitar obligar al desarrollador a eliminar el control antes de instalar la interfaz en el servidor, se ha dotado al control de la capacidad de detectar si la página en donde está alojado se ha servido desde la aplicación Node.js o no, y en el caso de que se encuentre alojada, el control se volverá invisible. De este modo, la interfaz se podrá instalar en el servidor sin la necesidad de acordarse de eliminar antes este elemento haciendo el proceso de despliegue aún más cómodo.

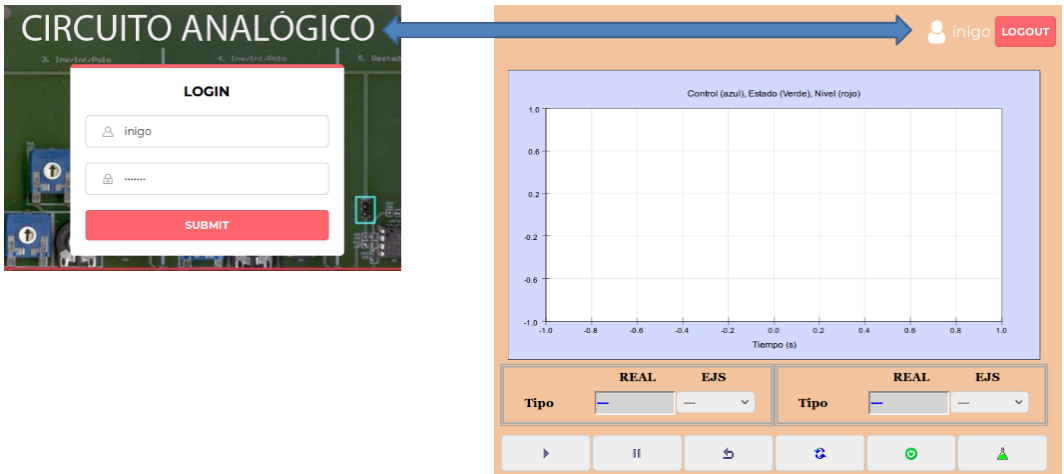


*Fig. 17. Página de Laboratorio con Autenticación Node.js*

El hecho de que el control sea innecesario en el servidor no desmerece en absoluto la importancia del mismo, ya que permite como se describirá con posterioridad, que se puedan realizar pruebas en local antes de instalar nuevas interfaces en el servidor, evitando tener que desplegar la página cada vez que se realice alguna modificación, reduciendo así los tiempos de parada de servicio de los laboratorios en mantenimiento.

Además, este control también capacita a un usuario avanzado a crear su propia página de interfaz para el laboratorio que desee, sin pon*er en ries*go la seguridad del sistema al imponer la necesidad de autenticación del usuario, al igual que si hubiera accedido al laboratorio utilizando la página de acceso de la propia aplicación Node.js. La figura 18 presenta una nueva interfaz para el laboratorio mostrado en la figura 17 utilizada directamente en un explorador de internet sin utilizar la aplicación Node.js.

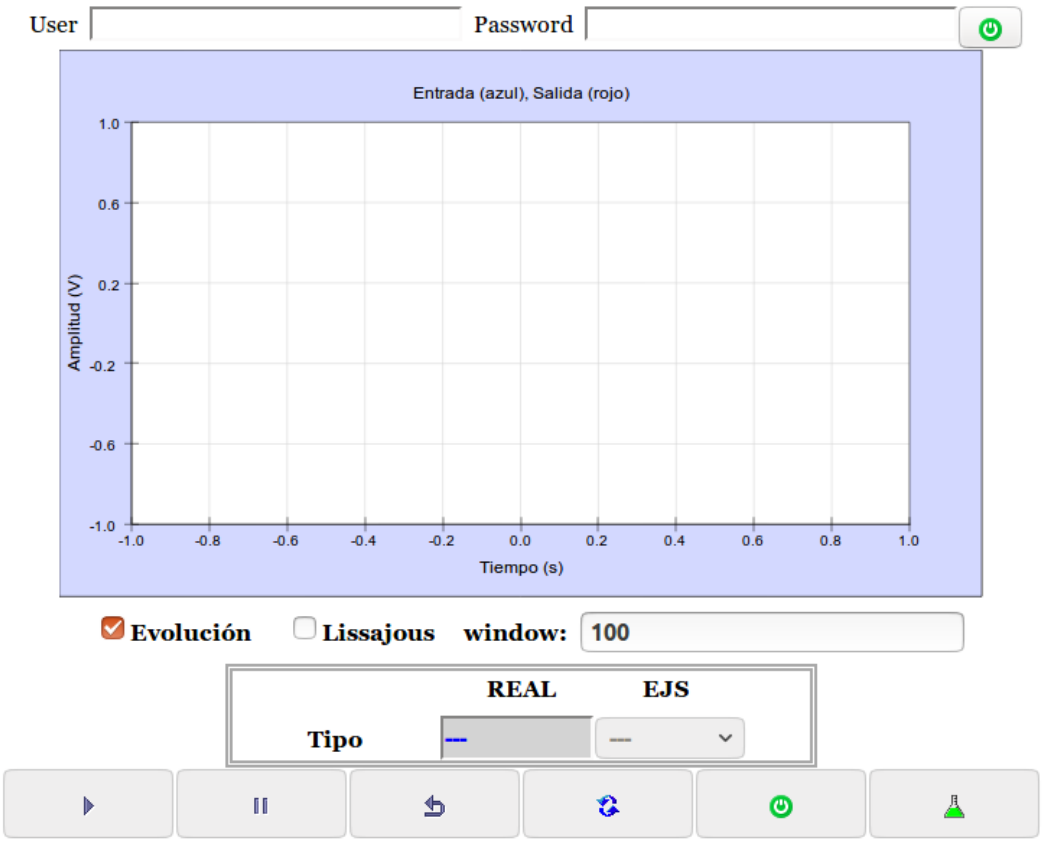


*Fig. 18. Página de Laboratorio sin Autenticación Node.js*

Todas estas características hacen de éste, un control esencial en el desarrollo de las interfaces de los laboratorios remotos de la UCM.

## 4.2.2 Control de Laboratorio

El control de laboratorio es un elemento visual compuesto por una barra de botones con funciones específicas para el manejo del laboratorio remoto al que se ha llamado *LabControl*. Esta barra de botones es común a todos los laboratorios remotos basados en ReNoLabs ya que está programada específicamente para interactuar con la aplicación Node.js instalada en cada uno de ellos.

El aspecto visual que presenta este control una vez añadido a la página es similar al mostrado en la figura 19.

*Fig. 19. Aspecto Visual de LabControl*

En ReNoLabs, el patrón de ejecución de los laboratorios remotos implementa una máquina de estados con la que se podrá interactuar por medio de esta barra. Estos estados y la función que realizan son las siguientes:

- *Start*: ejecuta las acciones necesarias para inicializar el controlador.
- *Ready*: estado de espera a recibir la señal de comienzo del experimento.
- *Running*: ejecución del bucle de retroalimentación del controlador.
- *Stop*: para la ejecución del bucle de retroalimentación del controlador.
- *Reset*: realiza las operaciones necesarias por cada laboratorio para poder reiniciar el experimento.
- *Ending*: ejecuta las acciones necesarias para finalizar el experimento.
- *Idle*: el laboratorio se encuentra en estado de reposo y no realiza ninguna acción.

En tiempo de ejecución del laboratorio, estos botones dispararán en el laboratorio remoto los eventos que mueven la máquina de estados. Los estados internos y los botones que disparan las transiciones se muestran en la figura 20 tomada de [9].

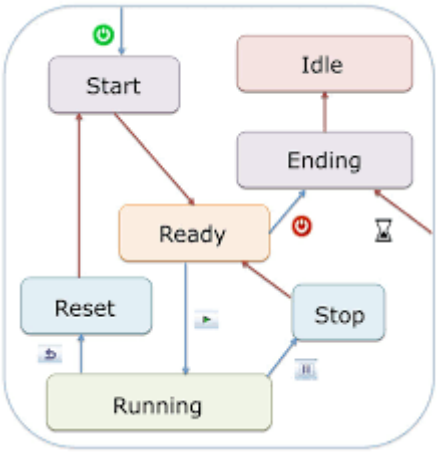


*Fig. 20. Máquina de Estados del Laboratorio ReNoLabs [9]*

Como puede apreciarse, la barra de botones mostrada en la figura 19 contiene un botón cuyo icono no aparece en ninguna de las transiciones del diagrama. Este botón es el que está posicionado a la derecha del todo. Como se describirá con posterioridad, como parte de este trabajo se ha implementado la posibilidad de que un usuario pueda implementar y desplegar su propio controlador de laboratorio, diferente al controlador dispuesto por el profesor. Este nuevo botón permitirá al alumno decidir previo al arranque del laboratorio, cuál de los dos controladores querrá utilizar durante la experiencia. Este controlador únicamente se permitirá cambiar mientras el laboratorio esté detenido por lo que si el alumno desea cambiarlo, deberá pararlo y arrancarlo de nuevo. En todo momento el alumno sabrá cuál de los controladores está utilizando ya que el icono mostrado por el botón mostrará un icono diferente para cada caso tal y como se muestra en la figura 21.

*Fig. 21. Botón Selector de Controlador*

El icono del matraz verde indica que se está utilizando el controlador del laboratorio mientras que el icono del usuario azul indicará que se está utilizando el controlador proporcionado por el alumno.

Para añadir este elemento a la interfaz, bastará con seleccionar el elemento dentro del grupo de controles visuales de ReNoLabs dentro de la vista HTML y arrastrarlo al árbol, dentro de cualquier elemento de tipo contenedor y asignarle un nombre único. A continuación allí podrán configurarse sus propiedades.

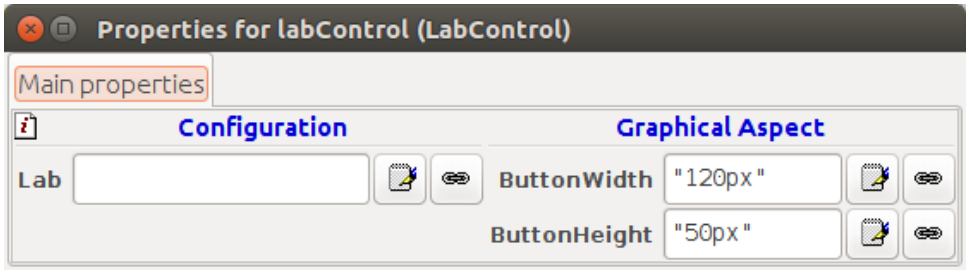


*Fig. 22. Propiedades de LabControl*

Las propiedades que este elemento expone son:

- *Lab*: establece el laboratorio al que está asociado de entre la lista de laboratorios añadidos al modelo a través de la extensión de modelo.
- *ButtonWidth*: establece la anchura de los botones.
- *ButtonHeight*: establece la altura de los botones.

Por cuestiones tanto de estética como de facilidad a la hora de configurar el elemento, la altura y la anchura de los cinco botones será la misma, consiguiendo establecer diez propiedades tan solo modificando las dos descritas.

Un aspecto importante a tener en cuenta acerca de este elemento es que, a diferencia de *LabLogin*, *LabControl* solamente puede estar asociado y controlar a un único laboratorio.

Otro aspecto fundamental es que esta barra solo funcionará cuando se haya realizado la autenticación en el laboratorio remoto bien mediante el control *LabLogin* o bien a través de la aplicación Node.js. Mientras esta autenticación no se realice, todos los botones permanecerán deshabilitados. Una vez realizada la autenticación, la barra recobrará su funcionalidad activando aquellos botones relevantes en el estado en el que se encuentra el laboratorio remoto.

En cuanto a la aportación de este elemento visual a ReNoLabs, podemos calificarlo de ventajoso ya que ahorra esfuerzo a la hora de crear una nueva interfaz para un laboratorio. Desde el punto de vista del número de elementos, la barra de control estaba compuesta por cinco botones que había que añadir manualmente, mientras que utilizando el nuevo elemento visual, esto mismo se consigue añadiendo un único elemento al árbol. Más aún, tomando en consideración las propiedades que se deben configurar, tenemos que para cada uno de los botones hay que definir al menos las dimensiones, el icono a mostrar y la acción a ejecutar al pulsar el botón. El nuevo elemento visual incorpora los iconos que se muestran sobre los botones y las acciones a realizar por cada uno de ellos, por lo que para tener una barra de control de laboratorio funcionando basta con añadir el nuevo elemento, establecer el laboratorio asociado y configurar las dimensiones de los botones una sola vez. Esto ahorra mucho tiempo y esfuerzo al profesor que está preparando la interfaz, dejándole más tiempo para dedicarse al laboratorio en sí que a su interfaz.

### 4.2.3 Selector de Función

Algunos laboratorios remotos disponen de entradas especiales cuya misión no es la de aportar un valor concreto a algún equipo de la planta, sino que es la de parametrizar de algún modo el comportamiento del propio laboratorio. Estas entradas en realidad funcionan a modo de selector de función de algún elemento del laboratorio y suelen ir acompañadas de una serie de valores que especifican los parámetros a aplicar para la función seleccionada. Como la explicación puede resultar un tanto confusa y difícil de visualizar, se ilustrará con un par de ejemplos.

Imaginemos que el laboratorio dispone de una fuente generadora de señales de diferentes formas de onda y que ésta está conectada a una de las señales de entrada a la planta. La fuente permite la selección automática de la forma de onda a generar y sus parámetros. Entre las señales que se pueden generar se encuentra la forma de onda senoidal y la señal en escalón. La forma de onda senoidal aceptará como parámetros la amplitud de onda y el periodo mientras que la señal escalón aceptará la amplitud del escalón y un parámetro temporal indicando el retardo en el cual se debe aplicar el mismo. El aspecto visual de este elemento sería similar al mostrado en la figura 23 en la que se muestra a la izquierda el componente cuando la función seleccionada es la senoidal con los parámetros de amplitud y periodo, y a la derecha cuando la función seleccionada es el escalón en el que los parámetros cambian a amplitud y retardo.

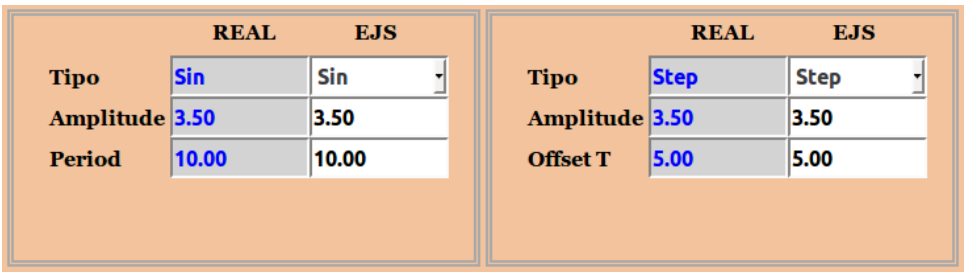


*Fig. 23. Selector de Forma de Onda*

En el siguiente ejemplo, supondremos que el software del controlador del laboratorio remoto está programado para admitir dos modos de funcionamiento. El controlador podrá funcionar tanto en modo controlador Proporcional, Integral y Derivativo (PID) como en modo controlador todo-nada (ON-OFF). Cuando el modo seleccionado es PID, los parámetros que aceptará serán las constantes proporcional, integral y derivativa, sin embargo, en el modo ON-OFF, únicamente aceptará un parámetro indicando el umbral en el que conmutará la señal de control. En este caso el aspecto visual en los casos en los que se selecciona el controlador PID y el controlador ON-OFF sería similar al mostrado en la figura 24.

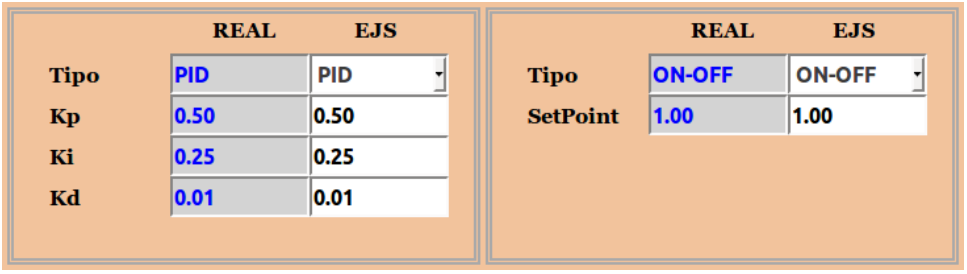


*Fig. 24. Selector de Tipo de Control*

Este elemento visual se ha implementado como parte del *Plugin* y se ha llamado *LabFunctionParameter*. Este elemento visual está compuesto fundamentalmente por un cuadro combinado que contiene la lista de funciones a seleccionar y una pila de cajas de texto, tantos como

número de parámetros que se pueden configurar, para introducir los valores que tomarán los mismos. Como se ha visto en los ejemplos, los parámetros aplicables a cada una de las funciones podrán variar, por lo que el elemento visual únicamente mostrará aquellos parámetros relevantes para la función seleccionada y ocultará el resto.

Tal y como está concebida la arquitectura de ReNoLabs, las señales de entrada al laboratorio remoto no se modifican en el momento en el que el usuario las cambia en la interfaz gráfica, sino cuando el usuario decide enviar los cambios a través de la barra de control de laboratorio vista en el apartado anterior. Por ello, es conveniente que la interfaz gráfica, además del valor que se quiere modificar muestre también en todo momento el valor que la señal tiene en el laboratorio remoto. De este modo, el usuario no pierde la perspectiva del estado real del laboratorio y puede observar si el laboratorio remoto ha aceptado sus cambios. Además, como ayuda visual, el fondo de los elementos cuyos valores difieran de los valores vigentes en el laboratorio remoto se volverán amarillos, de ese modo, al usuario no le costará identificar discrepancias entre el estado deseado y el real. En el momento en el que el valor introducido del parámetro coincida con el del laboratorio remoto, el fondo se volverá de nuevo blanco.

Para añadir este elemento a la interfaz, al igual que el control de laboratorio, bastará con seleccionar el elemento dentro del grupo de controles visuales de ReNoLabs dentro de la vista HTML y arrastrarlo al árbol, dentro de cualquier elemento de tipo contenedor, asignarle un nombre único y configurar sus propiedades.

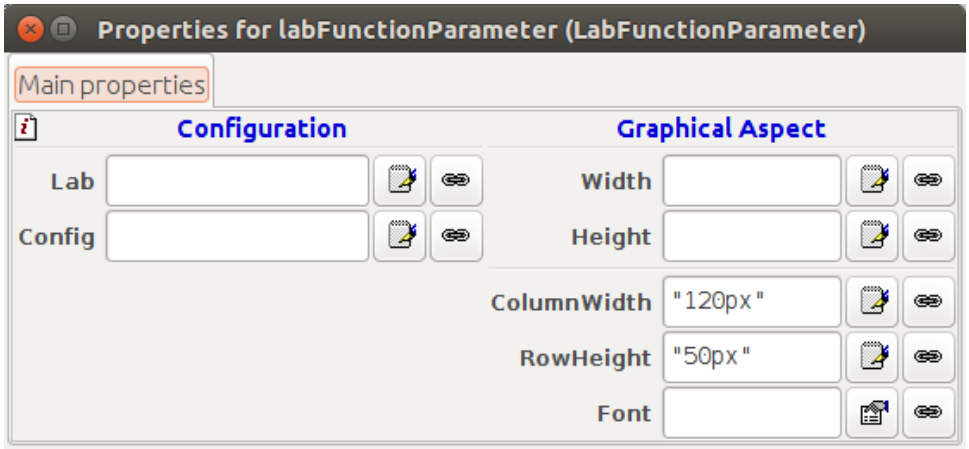


*Fig. 25. Propiedades de LabFunctionParameter*

Las propiedades que este elemento expone son:

- *Lab*: establece el laboratorio al que está asociado de entre la lista de laboratorios añadidos al modelo a través de la extensión de modelo.
- *Config*: establece el nombre de la entrada del laboratorio con la que se configurará el contenido visual del elemento.
- Width: establece la anchura completa del elemento.
- Height: establece la altura completa del elemento.
- ColumnWidth: establece la anchura de las columnas en donde se encuentran la etiqueta y los valores REAL y EJS.
- RowHeight: establece la altura de todas las filas de las que se componga el elemento.
- Font: establece el tipo y tamaño de fuente que utilizarán las etiquetas del elemento.

Al igual que ocurría con el control de laboratorio, el selector de función solamente puede estar asociado y controlar un único laboratorio.

Como puede observarse, ninguna de estas propiedades establece de forma directa ni las funciones que se pueden seleccionar ni los parámetros a configurar. De hecho, a la hora de añadir este componente al diseño de la página, el aspecto que éste mostrará en la vista previa será el mostrado en la figura 26, que no contiene información alguna.

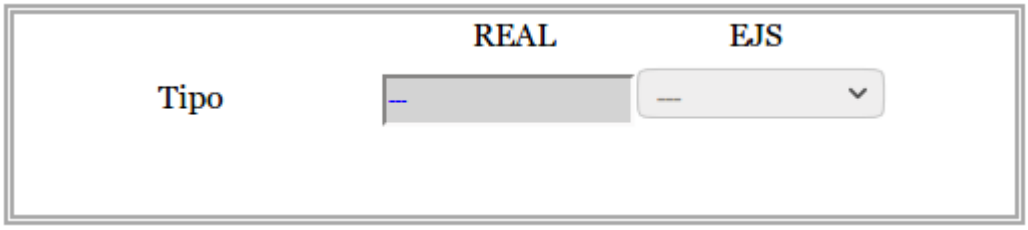


*Fig. 26. Aspecto Visual de LabFunctionParameter (Diseño)*

¿Cómo se pasa entonces de este aspecto en tiempo de diseño al componente final a la hora de visualizar la interfaz gráfica? Lo interesante de este elemento visual es que es capaz de auto-configurarse automáticamente en tiempo de ejecución con los parámetros necesarios para su correcta gestión, una vez autenticado en el laboratorio remoto. Esta información la recibe dinámicamente de la aplicación Node.js en forma de un objeto JSON con toda la información necesaria para su configuración. El parámetro *Config* será el que determine qué información debe recibir del servidor y una vez recibida, el objeto rellenará el cuadro combinado con la lista de opciones y añadirá tantas filas como sean necesarias para albergar los parámetros que pueden ser configurados. El objeto JSON también contiene información acerca de los parámetros utilizados por cada opción por lo que el elemento visual será capaz de mostrar y ocultar los parámetros en función de la selección haciendo que el aspecto final del elemento sea similar a los mostrados en los ejemplos.

Para comprender mejor el funcionamiento de este objeto JSON, se mostrará un ejemplo de configuración. Tomemos como ejemplo es siguiente objeto creado en la aplicación Node.js:

```
var config = {
    signal_1 : {
        parameter_names : ['Par 1', 'Par 2', 'Par 3'],
        options         : [{ name : 'Opt 1', parameter_indexes : [0, 1, 2]},
                           { name : 'Opt 2', parameter_indexes : [0, 1]},
                           { name : 'Opt 3', parameter_indexes : [1, 2]}
                          ]
    },
    signal_2 : {
        parameter_names : ['Par 1', 'Par 2'],
        options    : [{ name : 'Opt 1', parameter_indexes : []},
                      { name : 'Opt 2', parameter_indexes : [0, 1]}
                     ]
    }
};
```

El objeto ofrece dos señales llamadas “signal_1” y “signal_2”. Estos son los nombres de las señales con las que se configurarán los elementos *LabFunctionParameter*. en sus propiedades *Config*. El elemento configurado con la señas “signal_1” una vez conectado a la aplicación Node.js y recibido el objeto de configuración, rellenará el cuadro combinado con las tres opciones llamadas “Opt 1”, “Opt 2” y “Opt 3”. Al seleccionar “Opt 1”, la interfaz gráfica mostrará las cajas de texto correspondientes a los parámetros “Par 1”, “Par 2” y “Par 3”. Al seleccionar “Opt 2”, los parámetros que se mostrarán serán “Par 1” y “Par 3”. Finalmente, al seleccionar “Opt 3”, los parámetros cambiarán a “Par 2” y “Par 3”.

Si en lugar de seleccionar esta señal, seleccionamos la señal llamada “Signal 2”, el número de opciones con el que se rellenará el cuadro combinado será de dos en lugar de tres. En este caso, los parámetros se mostrará únicamente si se selecciona la opción “Opt 2” ya que la opción “Opt 1” no requiere ninguno.

El objeto de configuración de la aplicación Node.js puede contener cualquier número de señales y como se puede apreciar, para el correcto funcionamiento de este elemento es fundamental que el usuario se haya autenticado correctamente en el laboratorio remoto, por lo que el selector de función solo funcionará cuando la autenticación se haya realizado bien mediante el control *LabLogin* o bien a través de la aplicación Node.js. Cuando el usuario no está autenticado en el laboratorio, el aspecto que mostrará el elemento será el representado en tiempo de diseño y sus componentes aparecerán deshabilitados. Una vez realizada la autenticación, el selector de función se reconfigurará con los elementos necesarios, activará el cuadro combinado para permitir la selección de la función y mostrará los parámetros de la opción seleccionada por defecto.

De todos los elementos visuales implementados, éste es el que sin duda alguna ahorra más esfuerzo a un profesor que está implementando la interfaz gráfica para un laboratorio remoto con este tipo de casuística. Para comprender el ahorro de esfuerzo, retomemos el ejemplo del controlador que acepta los modos de control PID y ON-OFF descrito al inicio de este apartado.

Implementar este control utilizando el método tradicional, un profesor hubiera necesitado para comenzar, añadir al modelo 8 cajas de texto para mostrar los valores numéricos reales y propuestos de los 4 posibles parámetros, un cuadro combinado que almacenará la lista de opciones y otra caja de texto que muestre la opción vigente en el servidor. Además, también necesitaría añadir elementos auxiliares como etiquetas informativas y paneles para mantener el conjunto con la estética adecuada. Tras añadir los elementos gráficos tendría que configurar las propiedades como las dimensiones y fuente a utilizar de cada uno de ellos. Aún así esto no sería suficiente ya que faltaría configurar el comportamiento visual para modificar el color de fondo de aquellos parámetros en los que el valor real y el propuesto sea diferente y para mostrar u ocultar los elementos correspondientes a los parámetros en función de la opción seleccionada. En este escenario el profesor habría necesitado añadir y configurar del orden de 27 elementos en los que el cometer algún tipo de error resulta más que probable. Para empeorar más las cosas, como en cada laboratorio cada tipo de señal es diferente y puede tener diferentes parámetros, todo este esfuerzo de configuración se reduce al ámbito del laboratorio para el que se creó, disminuyendo las posibilidades de reutilización del componente.

Mediante el desarrollo de este elemento gráfico, realizado como un elemento genérico, todo este trabajo se ha reducido a la inclusión de un único elemento que internamente engloba todos los elementos mencionados anteriormente y que encierran el comportamiento preestablecido, y en la configuración de un conjunto mínimo de parámetros que afectan a todos los sub-elementos de los que se compone.

Por último, destacar que en este componente, se ha trasladado toda la complicación de configuración de la señal, del entorno gráfico a la aplicación Node.js. Sin embargo, al haberse implementado como un objeto JSON, esta configuración llega a ser trivial en el servidor.

## 4.3 Automatización del Despliegue y Mantenimiento

Para entender un poco mejor la solución implementada y los trabajos realizados en ReNoLabs en cuanto a la automatización de laboratorios remotos, se expondrá primeramente el detalle de cómo se estaba realizando y las dificultades encontradas a la hora de afrontar la automatización.

### 4.3.1 Proceso ReNoLabs

Cuando el profesor había terminado de implementar el diseño de la interfaz gráfica del laboratorio, procedía a su despliegue en el servidor. Para ello, primeramente generaba la simulación desde EjsS, mediante la acción que guarda el código generado junto a una serie de archivos auxiliares en un fichero comprimido. El archivo que contiene el modelo y la vista HTML se encuentra dentro de este fichero bajo el nombre finalizado en *"_Simulation.xhtml"*. Este archivo *xhtml* se extraía y se editaba manualmente para:

- Descomentar líneas de código que provocaban errores en la generación de código de EjsS. Estas líneas estaban relacionadas con llamadas a la librería *socket.io* y líneas de código EJS que debían ser interpretadas por el intérprete de EJS de la aplicación Node.js.
- Añadir a la página una cabecera en la que se muestra la información acerca del usuario conectado y el botón de desconexión existentes en las páginas de la aplicación web creada por ReNoLabs.
- Modificar la extensión del archivo *xhtml* por *ejs* para que pueda ser interpretado por la extensión *ejs* de la aplicación Node.js

Una vez realizado este proceso, el archivo con extensión *ejs* se copiaba en la carpeta apropiada del servidor quedando listo para su uso. La aplicación Node.js se encarga de servir las páginas web requeridas por los usuarios y una de esas páginas es precisamente la página generada en EjsS y editada manualmente con posterioridad.

### 4.3.2 Salvando Obstáculos

Vistos los pasos a seguir, quedaba de manifiesto que había varios obstáculos que había que salvar para lograr automatizar el proceso.

El primer obstáculo residía en la existencia de código EJS y llamadas a *socket.io* en la propia simulación de EjsS, código incompatible con la generación del archivo comprimido y el segundo era precisamente lo contrario, la necesidad de tener que añadir código, inexistente en la simulación.

Para afrontar estos obstáculos, se realizó un estudio minucioso del contenido de la página *ejs* que contiene la interfaz con el laboratorio para comprender mejor cuál era la aportación de las modificaciones manuales al proceso de despliegue. A grandes rasgos, esta página contiene los siguientes elementos:

- Cabeceras HTML que definen entre otras cosas los estilos visuales utilizados por los elementos la página.

- Un panel de navegación que muestra el nombre del usuario conectado con un enlace al menú principal de la aplicación Node.js y un botón que permite al usuario salir de la aplicación como se muestra en la figura 27.
- Una gran cantidad de código JavaScript encargado de construir y mostrar los elementos gráficos correspondientes al laboratorio y aplicar el comportamiento definido por el diseñador del mismo.

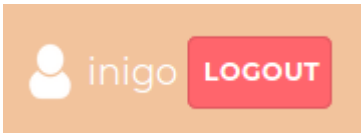


*Fig. 27. Panel de Navegación del Laboratorio*

De estos elementos, los añadidos durante el proceso manual son las cabeceras HTML con los estilos y el panel de navegación. El resto de contenido JavaScript lo aporta la generación de código de EjsS. Al estudiar diferentes laboratorios se comprobó que lo único que variaba entre un laboratorio y otro era el código JavaScript. Al ver este hecho, se realizó la prueba de separar el archivo *ejs* en dos. En un archivo con extensión *ejs* se puso la parte común de los laboratorios que contenía el código HTML y *ejs*, y en el segundo archivo con extensión *js* se puso únicamente el código JavaScript del laboratorio. Aprovechando una directiva del lenguaje *ejs* que permite incluir el contenido de un segundo archivo en tiempo de ejecución, se consiguió ejecutar exitosamente la interfaz del laboratorio del mismo modo que lo hacía el archivo original.

Por otro lado, EjsS tiene una característica que permite un tipo de generación en el que se separa el código HTML del código JavaScript encargado de gestionar el modelo y la vista, en dos archivos, uno con extensión *xhtml* y el otro con extensión *js*. Como en el servidor hemos separado el archivo de interfaz en dos, se verificó que para desplegar una nueva interfaz en un laboratorio era suficiente con sustituir el archivo *js* del servidor por el generado por EjsS, aunque este proceso también requería de la edición del archivo *js* para descomentar las líneas de código *socket.io* y *ejs*.

Para afrontar este último obstáculo, se estudió nuevamente la razón exacta para la existencia de este código para comprobar si se podría de algún modo eliminar o trasladar a otro lugar.

Las llamadas a *socket.io* se utilizan para las comunicaciones con la aplicación Node.js. Estas son las responsables de intercambiar información acerca del estado del laboratorio remoto y de enviar los comandos seleccionados por el usuario en la barra de control del laboratorio. Gracias al elemento de tipo laboratorio de la extensión de modelo *"Remote Lab"* implementada en el *Plugin*, este punto ha sido el más sencillo de solucionar ya que este elemento de modelo encapsula la gestión de las comunicaciones con Node.js haciendo uso de las llamadas a la librería *socket.io* de una manera totalmente transparente al usuario. Así, el código correspondiente a *socket.io* se puede eliminar completamente de las páginas de código del modelo de EjsS y por consiguiente, de la generación.

La razón para el uso de código EJS dentro de la simulación es un poco más compleja de describir. Sin entrar en excesivo detalle, el código EJS incluye una serie de variables calculadas por la aplicación Node.js que se encargan de garantizar que el laboratorio sea utilizado únicamente por usuarios autenticados. Ésta es una función sumamente importante y no se puede eliminar del archivo de simulación sin proporcionar una solución a la autenticación. Una vez más, gracias a la separación del archivo de laboratorio realizado en el servidor, el código *ejs* contenido en EjsS se pudo trasladar sin ningún problema al archivo fijo del servidor, permitiendo eliminarlo por completo del modelo de EjsS.

Liberando a EjsS de todo el código *socket.io* y EJS, se había logrado el objetivo de eliminar toda edición manual sobre los archivos generados por EjsS, y todo gracias a la separación de la página de laboratorio. Además, esta separación tiene un beneficio añadido. La parte EJS de la nueva página es la que aporta los estilos visuales y la estructura de navegación con la que se diseñó la aplicación Node.js por lo que, si en algún momento se decide realizar un cambio en el aspecto general de la aplicación y se necesita modificar bien los estilos o bien la zona de navegación, será suficiente con modificar este archivo sin afectar en absoluto al procedimiento de despliegue del laboratorio.

Todas estas modificaciones han permitido simplificar de un modo significativo el proceso de generación y despliegue del archivo generado en el servidor. Sin embargo, el propio proceso de copia del nuevo archivo *js* era algo que todavía tenía que hacer el profesor de forma manual. En el siguiente apartado se explica el modo en el que se ha llevado a cabo la automatización de este último paso.

### 4.3.3 Despliegue Automático

Una vez evitada la necesidad de cualquier procesado manual sobre el archivo generado por EjsS, la automatización del despliegue del laboratorio se convierte en una tarea relativamente simple. El despliegue del laboratorio se realizará mediante un nuevo botón que el *Plugin* añade a la barra de botones genérica. La figura 28 muestra el nuevo botón en el entorno EjsS ampliado.

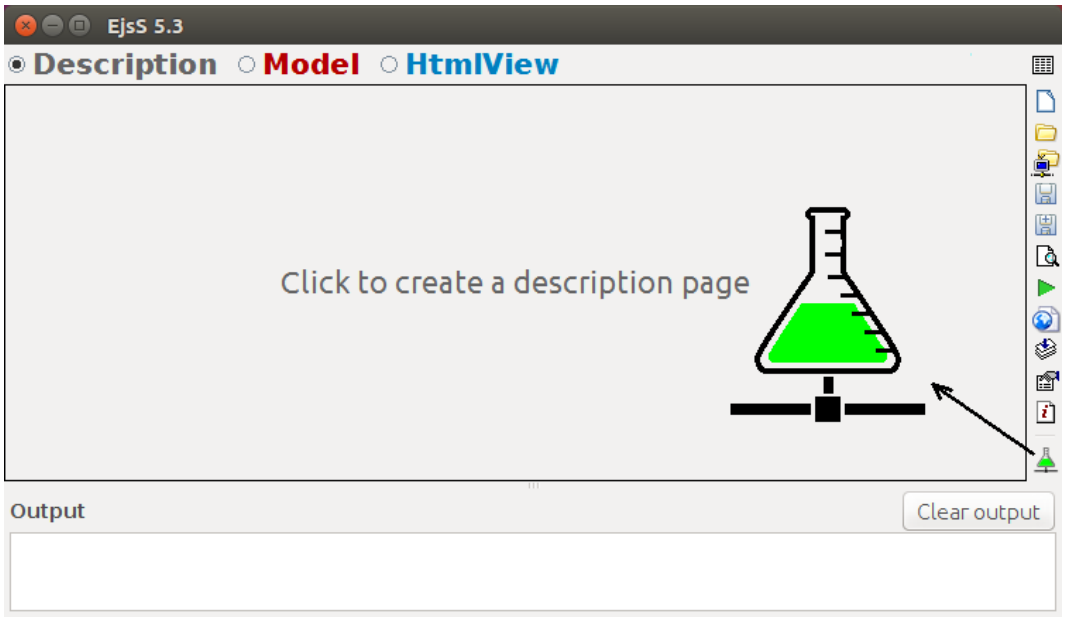


*Fig. 28. Extensión de Barra de Botones*

Una vez pulsado, el *Plugin* usará la generación interna de EjsS para crear el archivo *js* que contiene toda la información del modelo y la vista y tratará de enviarlo al servidor para su despliegue. Para ello, se pedirán al usuario la dirección y el puerto en donde está instalado el laboratorio así como sus credenciales. La figura 29 muestra los datos solicitados.

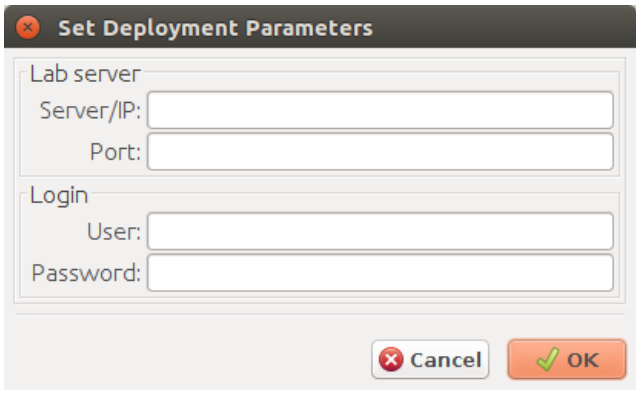


*Fig. 29. Solicitud de Credenciales para Despliegue de Laboratorio*

El *Plugin*, utilizando la librería *socket.io-java* se conectará al servidor indicado y se autenticará con las credenciales aportadas. En caso de éxito, el servidor aceptará el nuevo fichero enviado desde EjsS y sobrescribirá el existente. A partir de ese momento, todas las peticiones de conexión al laboratorio mostrarán el nuevo interfaz desplegado. En el caso de que las credenciales no sean correctas, se avisará al usuario y no se realizará ninguna modificación en el servidor.

Este nuevo mecanismo automatiza el despliegue de nuevos laboratorios ahorrando mucho tiempo al profesor, evitándole posibles errores que pueden surgir debido a la manipulación de los archivos intermedios durante el proceso de despliegue.

## 4.4 Versionado del Controlador

Como objetivo final se planteó aportar un mayor control sobre el software de control desplegado en el laboratorio. Este objetivo surgió para tratar de dar solución a un problema intrínseco al modo de trabajar utilizado hasta ahora. Al no existir un mecanismo automatizado de despliegue (ni del laboratorio, ni del software del controlador) se daba el caso de que existían diferentes copias del código de los controladores tanto en el servidor como en los diferentes discos duros que el profesorado había utilizado para su desarrollo. Mantener diferentes copias distribuidas de un software siempre ha sido un inconveniente allá donde exista software y en la UCM no ha sido diferente, en donde ha llegado a darse el caso de que se realizaban modificaciones del controlador sobre versiones que no correspondían con la versión del controlador cargado en el laboratorio, provocando errores no deseados en la ejecución del mismo, llegando a perder código funcional que ha tenido que ser implementado de nuevo.

Para tratar de aliviar este problema se pensó en la posibilidad de poder automatizar también el despliegue del software del controlador, permitiendo tanto descargar el código fuente como enviar uno nuevo. De ese modo el profesor podría estar siempre seguro de estar manteniendo el código verdaderamente desplegado en el laboratorio.

En la UCM disponen de laboratorios basados en diferentes plataformas y el software de controlador de cada una de ellas se ejecuta en un entorno diferente. Cada uno de estos entornos requiere un estudio que permita estudiar su viabilidad. En este trabajo se ha aportado una solución para las plataformas de la UCM basadas en Raspberry Pi.

Durante la realización de este trabajo se ha desarrollado un componente que permite conectarse al servidor de un laboratorio remoto y descargar el código fuente del controlador en uso para que pueda ser modificado y desplegado de nuevo en el laboratorio con los permisos adecuados. El componente permite además a cada usuario tener su propia copia de trabajo para mayor seguridad

antes de actualizar el software final. Este mecanismo posibilita un nuevo modo de trabajar que permite en todo momento mantener el código fuente en el propio laboratorio en donde se ejecuta evitando tener varias versiones distribuidas sin control en diferentes equipos. Este componente se ha implementado dentro del *Plugin* como una extensión de las opciones principales llamada *Controller*. La figura 30 muestra la interfaz gráfica de este nuevo componente.

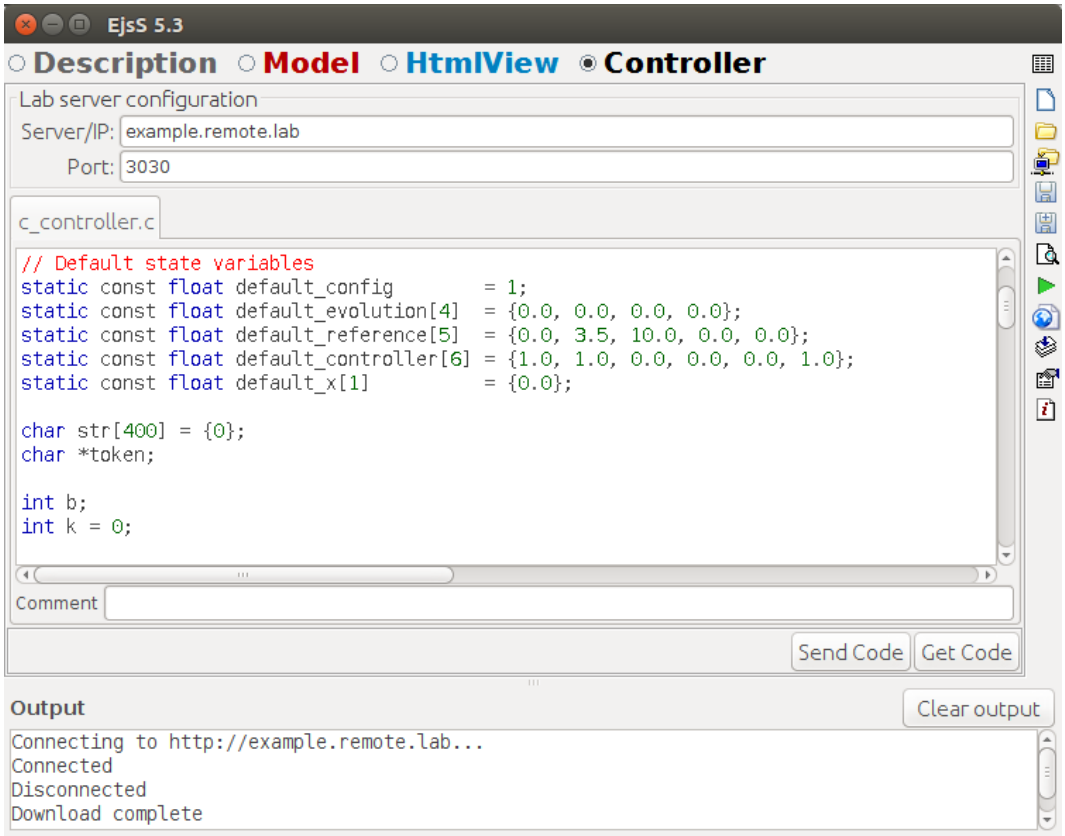


*Fig. 30. Extensión de Controlador*

Para conectarse al servidor y descargar el código fuente del controlador utilizando esta nueva extensión, el usuario deberá introducir la dirección y puerto del laboratorio en los cuadros de texto habilitados para ello y pulsar el botón *"Get Code"*. En ese momento se solicitarán las credenciales para conectarse al servidor y el controlador que se desea descargar, que puede ser el general o el personal. La figura 31 se muestran los datos solicitados.

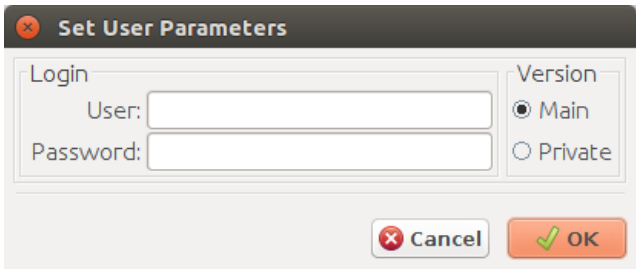


*Fig. 31. Solicitud de Credenciales para Manejo de Código de Controlador*

Al introducir un usuario y contraseña válidas, EjsS se conectará al servidor y descargará el código del controlador solicitado. Esto permite tanto a profesores como alumnos tener sus propias

versiones del controlador, ampliando significativamente la interacción que un alumno puede tener con el laboratorio remoto.

El código fuente puede estar compuesto de uno o varios archivos C por lo que EjsS descargará todos ellos mostrando cada uno de ellos en una pestaña independiente.

El usuario podrá modificar el código existente, agregar nuevos archivos o eliminar los ya existentes a su gusto. Una vez finalizada la edición, el usuario podrá desplegar su código para que pueda ser utilizado en sustitución al controlador general. Para ello el usuario pulsará el botón "Send Code". Nuevamente EjsS solicitará las credenciales del usuario para su despliegue, así como la versión del controlador a modificar al igual que se muestra en la figura 31. En el caso de seleccionar la versión general, el servidor comprobará que el usuario dispone los permisos para realizar esta operación, denegando la operación si carece de ellos. Una vez cargado el código en el servidor, se compilará y devolverá el resultado de la compilación al usuario. En el caso de que haya algún error de compilación el controlador existente no se verá afectado y el usuario tendrá la oportunidad de solucionar los problemas y volver a cargar el código sin errores. Una vez solucionados todo los errores de compilación, el controlador será reemplazado por el nuevo.

El lenguaje de programación elegido para la descarga y despliegue del controlador ha sido C ya que éste es el lenguaje utilizado en los laboratorios basados en Raspberry Pi. Sin embargo, existen trabajos en proceso en el marco de ReNoLabs que implementan los controladores en otros lenguajes de programación como Python y C++. Estos también podrían utilizarse siempre que implementen adecuadamente la interfaz software a través de los canales de entrada y salida estándar STDIO que requiere la aplicación Node.js. Para su uso se debería extender la funcionalidad de la aplicación Node.js para incluir por ejemplo, el comando de compilación de estos lenguajes en los casos de lenguajes compilados o realizar la acción que sea necesaria para el caso de lenguajes interpretados. Este tipo de labores se deja para trabajos futuros.

A diferencia de las extensiones anteriores pensadas para su uso en la interfaz gráfica generada, esta extensión a día de hoy únicamente está disponible para su uso desde EjsS. En el caso de que el profesor decida poner esta extensión del *Plugin* a disposición de terceros debe conocer que existen diversos aspectos con respecto a la seguridad que deberían controlarse. El permitir ejecutar código no controlado en un sistema siempre es potencialmente peligroso si se utiliza de una manera malintencionada.

## 4.5 Aplicación Node.js

Para dar soporte a todos estos desarrollos no ha sido suficiente con modificar el software en la parte del cliente sino que además, se han tenido que realizar diversas adaptaciones en la aplicación Node.js para soportarlas. Estas modificaciones realizadas a la aplicación Node.js son las que se han ido describiendo en los apartados anteriores dedicados al despliegue del laboratorio y que se listan a continuación:

- Se ha separado la página de laboratorio en un archivo *ejs* y otro *js* para permitir la modificación de la funcionalidad en el laboratorio desde el cliente manteniéndolo independiente del marco HTML diseñado para la aplicación Node.js.
- Se han añadido funciones para soportar la carga y descarga de software de controlador general y por usuario.
- Se han agregado permisos de usuario para controlar las modificaciones al software del controlador.

- Se han agregado permisos de usuario para permitir la conexión de usuarios en modo solo lectura (supervisores).
- Se han añadido objetos JSON que permiten la autoconfiguración de los elementos visuales destinados a representar entradas de función parametrizables.

Las modificaciones en el cliente junto con estas modificaciones en la aplicación Node.js han permitido afrontar exitosamente la automatización de los laboratorios remotos, ofreciendo, una base robusta sobre la que realizar futuros desarrollos.

## 4.6 Resumen de Elementos del *Plugin*

A continuación se enumerarán a modo de resumen, los nuevos elementos implementados en el *Plugin* para la gestión de laboratorios remotos desde EjsS:

- Modelo de Laboratorio: Se ha añadido un elemento de modelo que representa el estado del laboratorio completo, que encapsula las comunicaciones entre el cliente y el servidor.
- *LabLogin*: Elemento visual que permite la autenticación de un usuario en un laboratorio remoto utilizando el nombre y la contraseña.
- *LabControl*: Elemento visual compuesto de varios botones que permiten la interacción con el laboratorio remoto.
- *LabFunctionParameter*: Elemento visual configurable a través de un objeto JSON y que permite interactuar con las señales existentes en el laboratorios.
- Botón de despliegue: Botón añadido a la barra de botones principal que permite a un usuario de EjsS desplegar en el servidor una nueva interfaz de laboratorio.
- Pestaña *Controller*: Pestaña que permite a un usuario de EjsS, descargar, enviar y compilar en el servidor el código fuente del controlador de laboratorio.

# 5 Resultados Prácticos

Hasta el momento se ha descrito la necesidad de poder crear laboratorios remotos automáticamente, las herramientas utilizadas e incluso los componentes nuevos desarrollados, pero cabe preguntarse: ¿Cómo es verdaderamente un laboratorio remoto? ¿Qué se puede lograr mediante el uso de todos estos elementos? Existe un sinfín de laboratorios diferentes que pueden crearse con diferentes fines. En este trabajo nos hemos focalizado en fines educativos aunque es perfectamente extrapolable a otros ámbitos.

Con los conocimientos precisos acerca del laboratorio y usando las herramientas aquí mostradas, es posible transformar un laboratorio existente en un laboratorio que puede ser controlado de forma remota.

A continuación se mostrarán algunos ejemplos de laboratorios diseñados y desplegados con el *Plugin* desarrollado para EjsS y con la aplicación Node.js adaptada. Con ellos se mostrará de un modo práctico el uso de las diferentes extensiones implementadas y sus capacidades. Estos laboratorios son:

- Un laboratorio virtual.
- Un laboratorio real.
- Un laboratorio con controlador de usuario.
- Una conexión a múltiples laboratorios.

## 5.3 Laboratorio Virtual

Para comenzar con los ejemplos prácticos, se comenzará con algo sencillo y se probará el nuevo *Plugin* junto con la aplicación Node.js adaptada en un laboratorio virtual.

Supongamos que un profesor tiene en mente el desarrollo de un nuevo laboratorio para el control del nivel de agua en un tanque. La idea se encuentra en una fase muy primaria en la que todavía no ha concretado el objetivo final ni ha decidido los elementos físicos del laboratorio ni la plataforma en la que se ejecutará el controlador, pero le gustaría poder ver de alguna manera los resultados que obtendría con su idea inicial.

Para ayudar a este profesor se ha desarrollado un laboratorio remoto virtual que simula el control del nivel de agua en un tanque con el que podrá experimentar, y extraer conclusiones y nuevas ideas para evolucionar el laboratorio antes de proceder a su construcción. El laboratorio virtual que se ha modelado es el siguiente.

El laboratorio consta de un tanque de agua cuyo nivel se mide continuamente y se controla mediante un relé que activa o desactiva una bomba. Cuando la bomba está desconectada el nivel de agua disminuye a una velocidad constante y cuando está conectada aumenta también a una velocidad constante que puede ser igual o diferente a la velocidad de vaciado. La bomba actuará con un retardo desde la recepción de la señal de activación. El controlador activa la bomba cuando el nivel alcanza un determinado umbral y lo desconecta cuando llega a otro umbral. Estos umbrales podrán ser el mismo o diferentes.

El sistema está modelado en lazo cerrado como un autómata híbrido en el que se implementa un controlador de tipo ON-OFF. La figura 32 muestra el esquema de ejecución del sistema implementado.

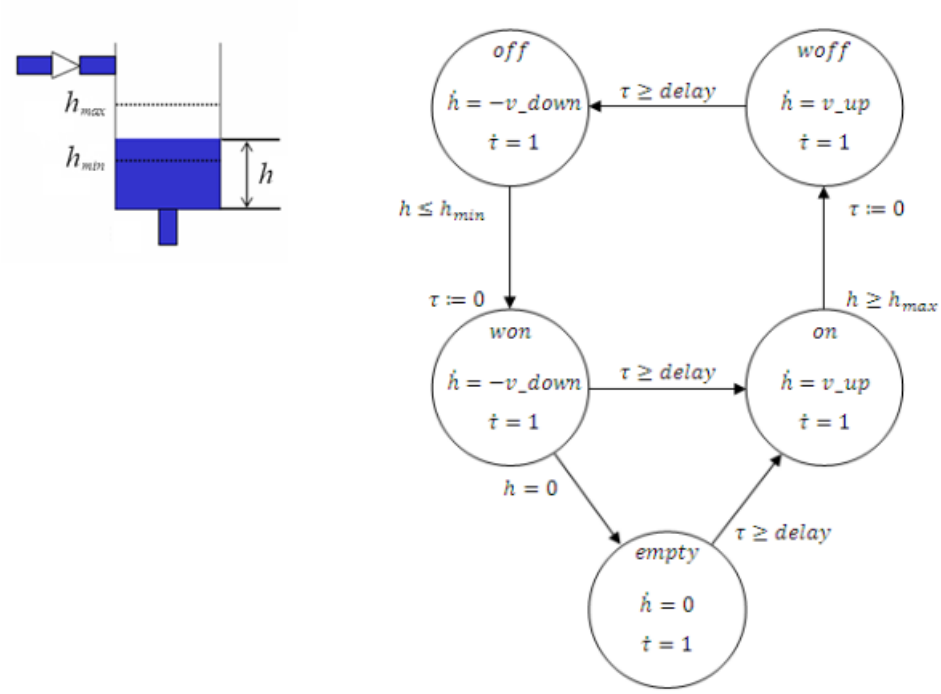


*Fig. 32. Autómata Híbrido*

El sistema debería permitir configurar no solo los umbrales de los niveles de agua que conmutan la señal de control, sino también otros parámetros de funcionamiento de la propia simulación. La lista completa de parámetros es la siguiente:

- Controlador:
  - Umbral Máximo
  - Umbral Mínimo
- Simulación:
  - Velocidad de llenado (en cm/s)
  - Velocidad de vaciado (en cm/s)
  - Retardo de acción de la bomba (en s)

Además, el laboratorio deberá aportar como salidas la siguiente información:

- Señal de control (ON, OFF)
- Estado interno del autómata
- Nivel de agua (en cm)
- Referencia temporal (en segundos)

En todas las simulaciones se supondrá que el tanque se encuentra inicialmente vacío.

Para este ejemplo, la aplicación Node.js se ha instalado en una Raspberry Pi que hará de servidor y el software del laboratorio virtual se ha implementado en C, aunque podría haberse elegido cualquier otra plataforma que soporte Node.js y otro lenguaje de programación. La aplicación

Node.js se comunica con el software del laboratorio a través de los canales estándar STDIO y la página de laboratorio inicialmente no tiene contenido.

La interfaz gráfica y el despliegue en el servidor de este laboratorio se han realizado utilizando el nuevo *Plugin* de EjsS.

Para comenzar el desarrollo de la nueva interfaz, haciendo uso de la extensión del modelo, se ha añadido un elemento de tipo *"Remote Lab"* en el cual se basarán los elementos visuales añadidos a la interfaz gráfica. La ejecución del laboratorio se controlará utilizando el elemento *LabControl*, el cual se asociará al elemento añadido al modelo.

Como se puede observar, se puede considerar que el controlador tiene dos modos de funcionamiento: modo de umbral único y modo de doble umbral. En el modo de umbral único, el controlador utilizará el mismo valor para la conmutación de la señal de control. En el modo de doble umbral, el controlador utilizará valores diferentes para disparar la conmutación de la señal de control. En el fondo, el modo de umbral único no es otro que el de doble umbral en el que el umbral máximo y mínimo son el mismo. Sin embargo, desde el punto de vista de la interfaz, resulta más cómodo para un usuario introducir un único valor en lugar de tener que repetirlo. Para posibilitar esta configuración se ha programado en el controlador de laboratorio una señal llamada "controller" y se ha utilizado en la vista HTML un elemento *LabFunctionParameter* asociado a la señal. Para su configuración en la aplicación Node.js se ha utilizado el siguiente objeto JSON:

```
controller : {
    parameter_names : ['Threshold', 'Min', 'Max'],
    options         : [{ name : 'Single', parameter_indexes : [0]},
                       { name : 'Double', parameter_indexes : [1, 2]}
                      ]
}
```

Para permitir al usuario introducir los valores de la simulación, bien se podrían haber añadido a la interfaz tres cajas de texto que corresponden con la configuración de las velocidades de llenado, vaciado y tiempo de retardo de la bomba, sin embargo, nuevamente vamos a sacar partido del elemento *LabFunctionParameter*. Este componente, además de añadir los elementos visuales necesarios, se encarga de enviar los parámetros al laboratorio y gestionar los detalles de la interfaz gráfica sin que tengamos que añadir ninguna línea de código. Esta señal se ha llamado "simulation" en el controlador de laboratorio y el objeto JSON utilizado en la aplicación Node.js en este caso ha sido:

```
simulation : {
    parameter_names : ['Vup', 'Vdown', 'Delay'],
    options         : [{ name : 'Config', parameter_indexes : [0, 1, 2]}]
}
```

En este caso, puede resultar poco natural utilizar este elemento ya que los parámetros a configurar son siempre los mismos y el cuadro combinado estará relleno con un único valor, pero teniendo en cuenta el esfuerzo ahorrado, este detalle pasa a un segundo plano.

Las salidas que se mostrarán serán la señal de control (0 = OFF, 1 = ON), el estado interno del autómata (0 = EMPTY, 1 = ON, 2 = WOFF, 3 = OFF, 4 = ON) y el nivel de agua en cm, todas ellas dispuestas en una única grafica temporal por simplicidad.

La interfaz desarrollada con estos elementos se ha desplegado en el laboratorio utilizando el botón añadido por el *Plugin*. El resultado se muestra en la figura 33.


*Fig. 33. Interfaz Laboratorio Tanque*

Como el laboratorio todavía no ha entrado en producción y no dispone aún de una página, no nos preocupa el tiempo de "parada de servicio" del laboratorio por lo que en este ejemplo, las pruebas se han realizado directamente en el servidor, prescindiendo del elemento *LabLogin*. Si en un futuro el laboratorio entrase en producción y fuera puesto a disposición de los alumnos, sería muy conveniente añadir este elemento que permitiría realizar pruebas sobre el mismo minimizando los tiempos de corte de servicio. Una vez desplegada la página de laboratorio se ha procedido a realizar varias pruebas para verificar su correcto funcionamiento.

En la figura 33 se muestra un ejemplo de ejecución utilizando unos parámetros de configuración y un controlador con doble umbral. Esto se aprecia en el cuadro combinado de la derecha cuyo valor seleccionado es *"Double"* y aparecen dos parámetros mínimo y máximo que se pueden configurar.


*Fig. 34. Interfaz Laboratorio Tanque Umbral Único*

En la figura 34 puede verse otro ejemplo en el que se ha utilizado un umbral único para los valores del controlador. Se puede apreciar que en el cuadro combinado de la derecha el valor seleccionado es *"Single"* y aparece un único valor en la lista de parámetros con los que se configura.

En los ejemplos mostrados, el tanque no llega a vaciarse. Esto puede observarse ya que la línea verde nunca toma el valor cero. Para probar este estado, se han modificado ligeramente los parámetros tal y como se muestra en la figura 35. En esta figura puede observarse que durante unos instantes, el tanque llega a estar vacío ya que la línea verde alcanza el valor cero, asociado al estado vacío del autómata.


*Fig. 35. Interfaz Laboratorio Tanque con Estado Vacío*

Para concluir con los resultados del laboratorio virtual y tan solo a modo informativo, cabe destacar que el tiempo empleado en la implementación y resolución de errores del mismo ha sido del orden de 5 horas, mientras que la implementación y despliegue de la interfaz gráfica con las facilidades implementadas en este trabajo, ha requerido tan solo 15 minutos. Esto pone de manifiesto que la mayor parte del esfuerzo en la implantación completa del nuevo laboratorio ha caído en la parte en la que el profesor debe poder dedicarle más esfuerzo, facilitándole enormemente otras tareas que, aunque necesarias, son ajenas al objetivo puramente docente del laboratorio.

## 5.1 Laboratorio Real

Una vez vistos los resultados obtenidos contra el laboratorio virtual, se ha probado a realizar la misma actividad pero en esta ocasión contra un laboratorio real existente en la UCM. El laboratorio que se ha seleccionado es el correspondiente a la identificación de la respuesta temporal de un sistema propuesto en la asignatura de Sistemas Lineales en el grado de Ingeniería Electrónica de Comunicaciones durante el curso 2017/18.

El objetivo de esta práctica es la obtención de la función de transferencia asociada a los modelos de diferentes sistemas, a partir del estudio de su respuesta temporal frente a una entrada escalón. A diferencia del laboratorio anterior en el que no existían elementos físicos, en este laboratorio, se utiliza un circuito real reconfigurable formado por un conjunto de amplificadores operacionales, resistencias variables, condensadores y jumpers que permiten utilizar el circuito de múltiples formas y destinarlo a diferentes prácticas.

El circuito consta de cuatro elementos “Inv/Int/Polos” (elementos que permiten configurar un amplificador operacional como una ganancia inversora, como un integrador inversor o como un polo con una ganancia negativa), dos restadores y un controlador (configurable en modo P, I, PI, PD, PID, red adelanto, red atraso, o red de atraso y adelanto) dispuestos en serie. Estos elementos pueden incluirse o excluirse del experimento mediante una serie de jumpers dispuestos de forma estratégica. La figura 36 muestra el circuito junto con su esquema eléctrico.

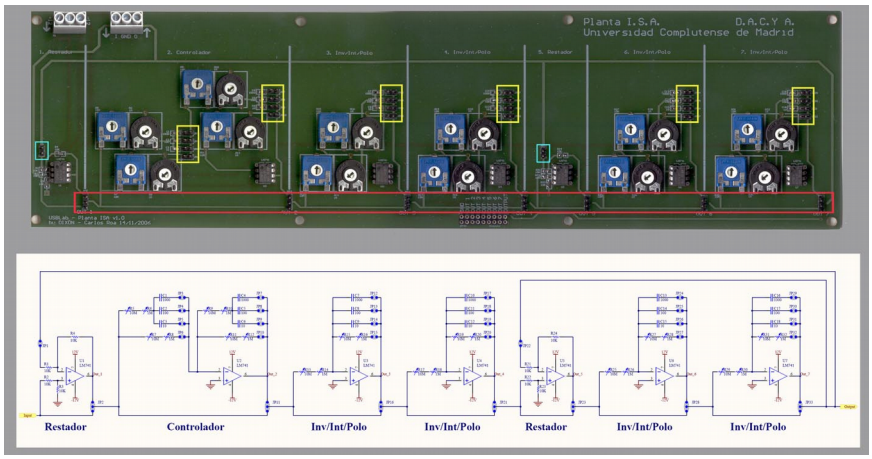


*Fig. 36. Circuito Analógico Configurable*

Al igual que se ha hecho con el laboratorio virtual, la nueva aplicación Node.js se ha instalado en la Raspberry Pi sobre la que estaba construido el laboratorio.

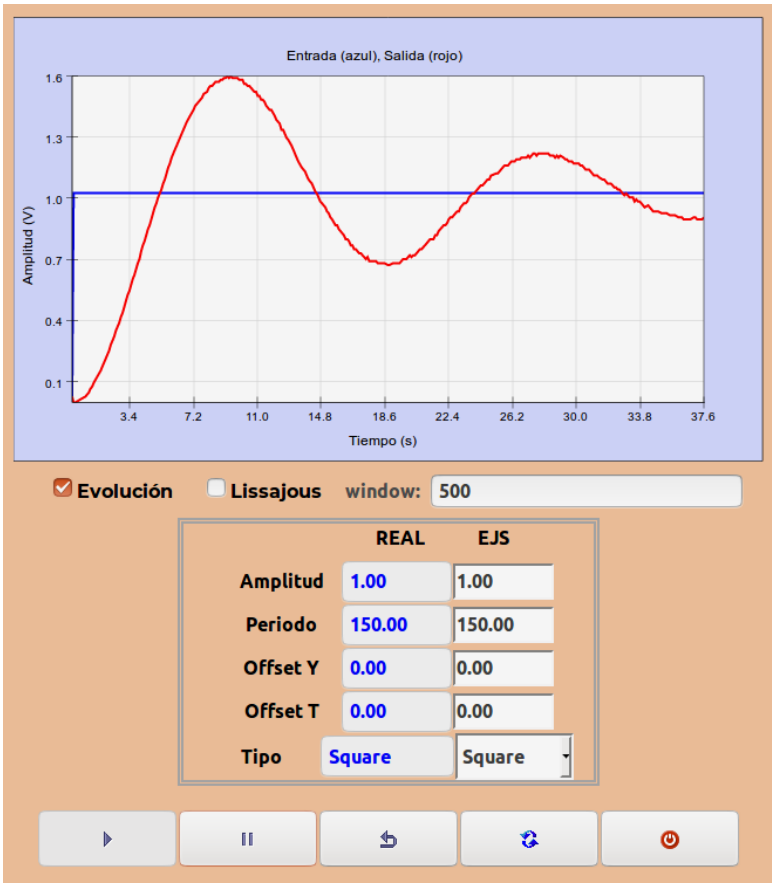


*Fig. 37. Laboratorio Original Circuito Analógico Configurable*

En este caso, inicialmente ya existe una página de laboratorio, mostrada en la figura 37, creada con la herramienta EjsS con el método tradicional manual descrito en el capítulo anterior. Esta página permite configurar la señal de referencia de entrada al laboratorio con las funciones: Seno, Cuadrada, Triangular, Impulso, Escalón y una referencia externa.

Como salida, el laboratorio devolverá la respuesta del sistema a la función de referencia establecida. La página diseñada, permite ver la salida tanto en una gráfica temporal como en una gráfica Lissajous, que muestra la respuesta de salida frente a su entrada.

En esta ocasión, el software del laboratorio virtual no se ha modificado y se ha utilizado el que ya estaba disponible. El controlador de laboratorio en este caso también está implementado en C.

Para este laboratorio se ha creado una nueva página replicando la ya existente utilizando el Plugin de EjsS. Igualmente, el despliegue de la misma se ha realizado utilizando la funcionalidad de despliegue del *Plugin*.

Como siempre, el primer paso para la creación de una página de laboratorio será añadir a la simulación EjsS, el elemento de tipo *"Remote Lab"* que soportará la conexión con el laboratorio. Para gestionar la ejecución del laboratorio, la interfaz dispone de los botones característicos de los laboratorios de ReNoLabs. Estos botones se han añadido agregando a la interfaz gráfica el elemento *LabControl*, asociado al elemento de modelo de laboratorio añadido.

Para permitir al alumno seleccionar la función de referencia se ha aprovechado la señal llamada "reference" existente en el controlador de laboratorio y para ello se ha añadido un elemento *LabFunctionParameter* asociado a ella. En la aplicación Node.js, esta señal se ha configurado con el siguiente objeto JSON:

```
reference : {
    parameter_names : ['Amplitude', 'Period', 'Offset Y', 'Offset T'],
    options    : [{ name : 'Sin', parameter_indexes : [0, 1, 2, 3]},
                  { name : 'Square', parameter_indexes : [0, 1, 2, 3]},
                  { name : 'Triangular', parameter_indexes : [0, 1, 2, 3]},
                  { name : 'Impulse', parameter_indexes : [0, 1, 2, 3]},
                  { name : 'Step', parameter_indexes : [0]},
                  { name : 'Extern', parameter_indexes : []}
                 ]
}
```

Para representar las salidas, se han mantenido las gráficas temporal y Lissajous existentes.

En este caso, al tratarse de un laboratorio en producción, interesa realizar todas las pruebas de la interfaz gráfica en local antes de modificar definitivamente la instalada en el servidor. Para poder realizar esto, se ha añadido un elemento *LabLogin* a la interfaz de usuario asociado al elemento de modelo de laboratorio añadido.

Para realizar las pruebas locales se debe hacer uso del proceso de generación provisto por EjsS, en el que genera la simulación en un archivo comprimido. De este archivo comprimido, se extrae el fichero llamado *"*_Simulation.xhtml"* y se ejecuta directamente sobre un explorador.

La interfaz gráfica resultante se muestra en la figura 38. En esta figura se puede apreciar que aparece en la parte superior el elemento *LabLogin* que nos permitirá conectarnos al laboratorio utilizando las credenciales de usuario y contraseña. Esta página, al no estar integrada en la

aplicación Node.js, no tiene la apariencia visual que aportan los estilos de la aplicación, no obstante, una vez autenticados en el servidor, la página es plenamente funcional.

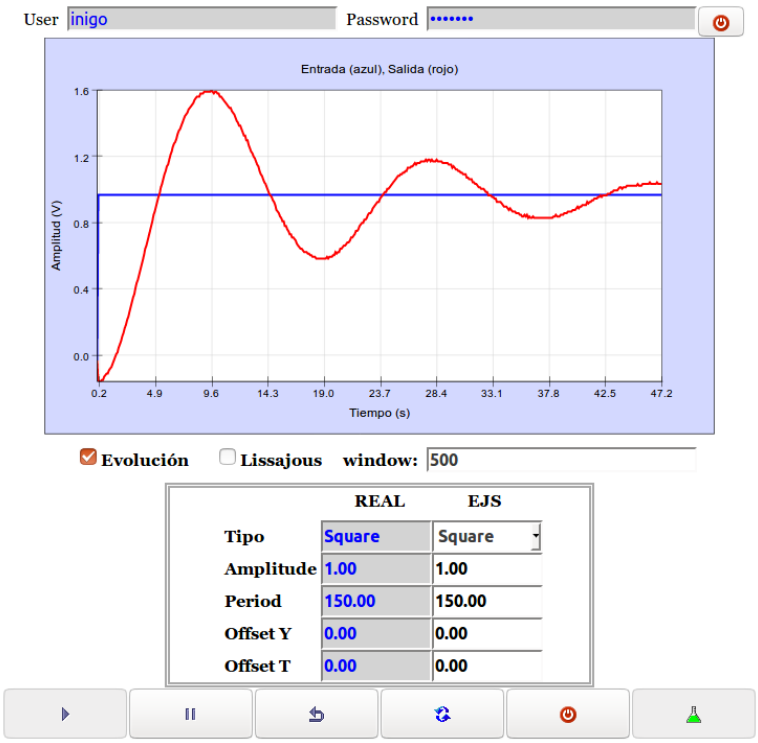


*Fig. 38. Nueva Interfaz Laboratorio Circuito Analógico (Test Local)*

Tras realizar unas pruebas preliminares se comprueba el correcto funcionamiento de la página. La propia figura 38 muestra la ejecución del laboratorio con los mismos parámetros utilizados en la figura 37 en la que se aprecian los mismos resultados.

Durante las pruebas, hemos sido capaces de manejar el laboratorio con la nueva interfaz sin parar el servicio del laboratorio. Como es natural, al estar conectados como usuarios, estamos impidiendo que otro alumno se pueda conectar durante el tiempo que duran las pruebas, al igual que ocurre cuando hay otro alumno conectado. Pero en el momento en el que realizamos la desconexión, el alumno puede conectarse libremente. Lo interesante es que al no haber procedido al despliegue de la nueva interfaz, lo que visualizará el alumno será todavía la interfaz gráfica original.

Una vez que estamos satisfechos con los resultados, la nueva interfaz se despliega al servidor usando el nuevo botón de EjsS añadido por el *Plugin*. Con esta interfaz, el alumno podrá manejar el laboratorio utilizando las mismas señales que en la interfaz original. La interfaz desplegada tomará los estilos visuales de la aplicación Node.js cuando se sirve desde el servidor. La figura 39 muestra el resultado final de la nueva interfaz gráfica servida al alumno a través de la aplicación Node.js. Como puede apreciarse, el control *LabLogin* utilizado durante las pruebas no se muestra en la página ya que ha detectado que se está sirviendo desde la aplicación Node.js, en la que el alumno ya ha sido previamente autenticado. En su lugar aparece la cabecera suministrada por la aplicación para permitir al usuario volver al menú principal y desconectarse.

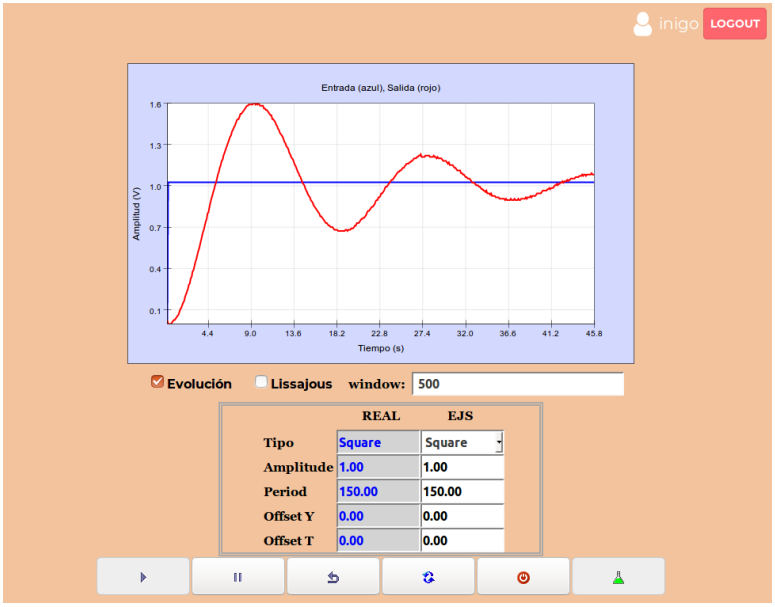


*Fig. 39. Nueva Interfaz Laboratorio Circuito Analógico (Desplegado)*

En la figura también se aprecia que los resultados obtenidos desde el servidor corresponden con los obtenidos durante las pruebas en local.

Se ha comprobado que el hecho de que el laboratorio ser real o virtual es un dato totalmente transparente para la aplicación Node.js y que por lo tanto, el *Plugin* y las adaptaciones a la aplicación Node.js son válidas para ambos tipos de laboratorio remoto.

## 5.2 Laboratorio con Controlador de Usuario

Hasta ahora se han visto los resultados de utilizar el *Plugin* para la generación gráfica de la página del laboratorio y el despliegue de la misma en el servidor. Sin embargo, el *Plugin* ofrece otra funcionalidad muy importante destinada al despliegue del código del controlador del laboratorio. Esta funcionalidad se ha utilizado para sustituir el controlador del laboratorio real descrito en el apartado anterior.

La modificación que se ha planteado para el controlador ha sido una muy sencilla y con el único fin de demostrar las capacidades de la extensión de controlador implementada en el *Plugin*. Una vez demostrada la capacidad de alterar el código a ejecutar en el controlador de laboratorio, las posibilidades que ofrecería esta herramienta serían muy grandes.

La modificación que se realizará sobre el controlador de laboratorio será la de añadir una nueva forma de onda a las entradas existentes. El laboratorio actual admite las señales con forma senoidal, cuadrada, triangular, impulso, escalón y referencia externa. A estas formas de onda se añadirá una nueva correspondiente a una onda de sierra como la mostrada en la figura 40.

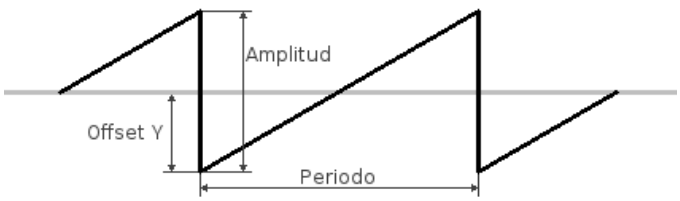


*Fig. 40. Onda de Sierra*

Al seleccionar esta forma de onda se podrán configurar la amplitud, el periodo y el desplazamiento vertical, representados en la propia figura. Inicialmente, el código fuente del controlador se descargó del servidor utilizando la extensión *"Controller"* implementada en el *Plugin*. El software del controlador está formado por 2 ficheros C que son *circuit_practice_controller_10.c* y *getboardISA.c*. Para añadir esta nueva forma de onda se modificó el fichero *circuit_practice_controller_10.c* directamente en el entorno EjsS tal y como se muestra en la figura 41.

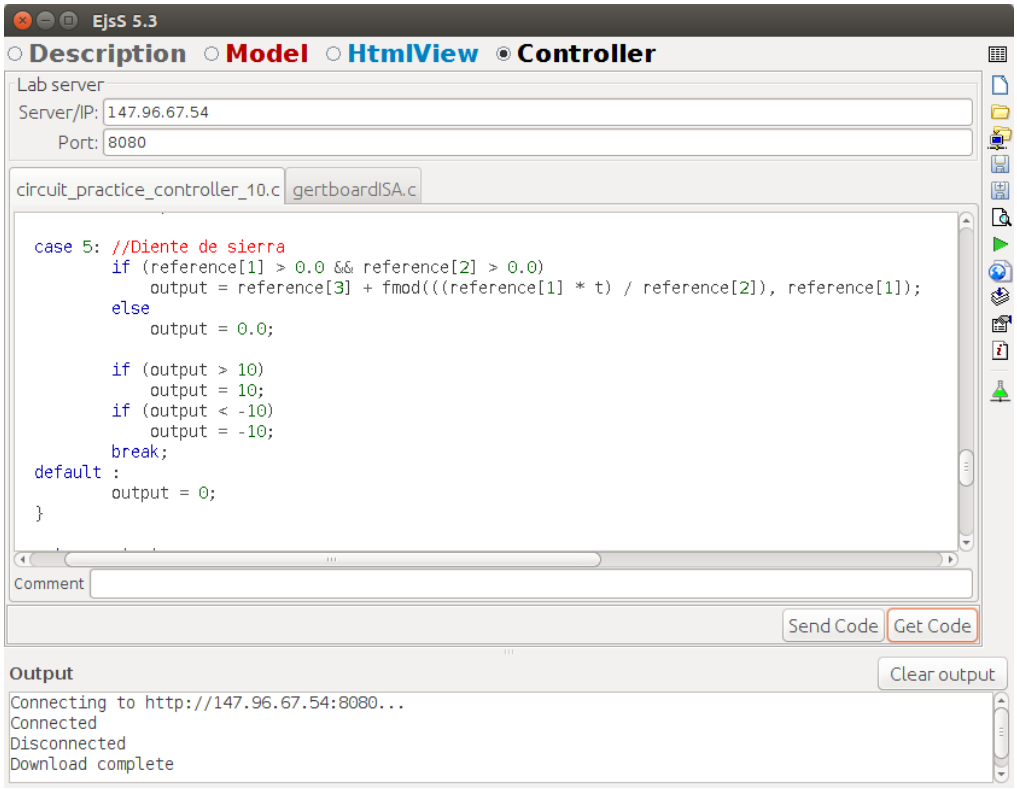


*Fig. 41. Controlador de Laboratorio en EjsS*

Una vez añadida la función, se desplegó el nuevo controlador en el servidor, como controlador general, utilizando para ello un usuario con permisos suficientes para modificar este controlador.

Además de esta modificación, se ha tenido que modificar el objeto JSON "reference" para añadir esta nueva opción junto con sus parámetros. El objeto JSON ha quedado de la siguiente forma:

```
reference : {
    parameter_names : ['Amplitude', 'Period', 'Offset Y', 'Offset T'],
    options    : [{ name : 'Sin', parameter_indexes : [0, 1, 2, 3]},
                  { name : 'Square', parameter_indexes : [0, 1, 2, 3]},
                  { name : 'Triangular', parameter_indexes : [0, 1, 2, 3]},
                  { name : 'Impulse', parameter_indexes : [0, 1, 2, 3]},
                  { name : 'Step', parameter_indexes : [0]},
                  { name : 'Sawtooth', parameter_indexes : [0, 1, 2]},
                  { name : 'Extern', parameter_indexes : []}
                 ]
}
```

Una vez desplegado, se procedió a probarlo en la página de laboratorio instalada en el servidor. Al poner el laboratorio en marcha, se ha comprobado que efectivamente, el controlador de laboratorio dispone de una nueva función de referencia. La figura 42 muestra la nueva opción implementada.

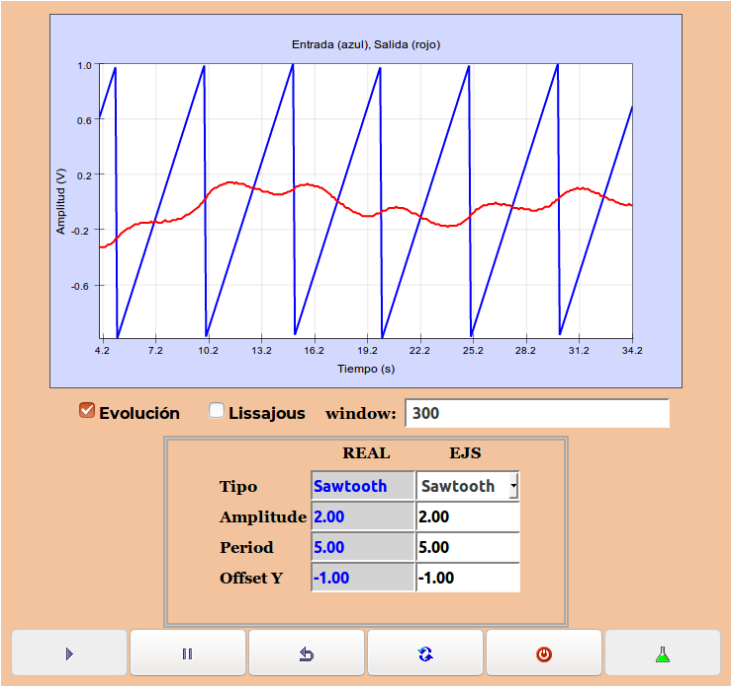


*Fig. 42. Onda de Sierra en Controlador Principal*

La extensión permite el despliegue del controlador de laboratorio como controlador personal en lugar de sobrescribir el controlador general. Para demostrar esta capacidad, se ha tomado el controlador con la nueva señal de referencia de onda de sierra y se ha modificado desde EjsS para que sea una onda de sierra invertida. Este nuevo controlador se ha desplegado como controlador de usuario en lugar de como controlador general y se ha probado el resultado en el servidor.

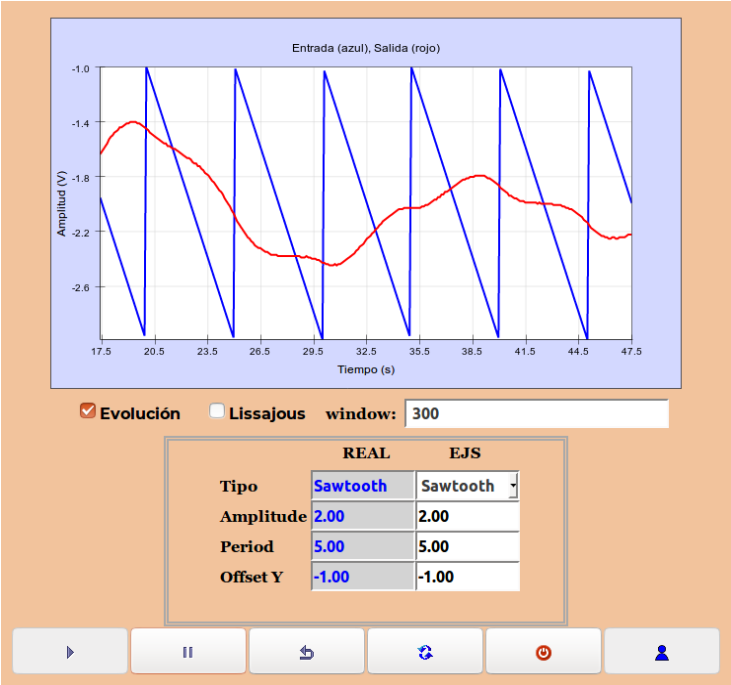


*Fig. 43. Onda de Sierra Invertida en Controlador de Alumno*

Para ello, nos hemos conectado con las credenciales del usuario utilizado para el despliegue del controlador personalizado y hemos seleccionado en la página que se quería hacer uso del este nuevo controlador. La figura 43 muestra el resultado obtenido. En ella se aprecia que la señal de onda de sierra está invertida con respecto a la mostrada en la figura 42. En el momento en el que el usuario selecciona el controlador general, la onda de sierra vuelve a ser la original.

Finalmente, para comprobar el correcto funcionamiento de los permisos de usuario para poder estar seguros de que un usuario no pueda modificar accidentalmente el controlador general, se ha tratado de desplegar el controlador de onda de sierra invertido como controlador general por un usuario sin permisos. El resultado ha sido un mensaje de error en la salida de EjsS indicando que no ha sido posible su despliegue debido a que el usuario carece de los permisos necesarios tal y como se muestra en la figura 44.


*Fig. 44. Despliegue de Controlador de Alumno sin Permisos*

Para comprobar que efectivamente el controlador no se ha visto modificado, nos hemos conectado al laboratorio con un usuario y hemos arrancado el controlador general. La onda de sierra mostrada en las gráficas ha sido la esperada, que es la original.

Todas estas pruebas demuestran el correcto funcionamiento de esta extensión. Como se ha comentado, esta modificación ha sido muy simple pero demuestra que es una capacidad con mucho potencial, tanto para el profesor como para los alumnos que quieran practicar con sus propias implementaciones.

## 5.4 Conexión a Múltiples Laboratorios

Para concluir con los ejemplos prácticos, se demostrará la capacidad de conexión simultánea a múltiples laboratorios remotos. A pesar de que esta capacidad permitiría a un usuario trabajar en más de un laboratorio remoto a la vez, la utilidad práctica de este escenario está aún sin explorar. Cuando se desarrolló esta funcionalidad, se pensó en un caso práctico más útil como es el de poder supervisar o monitorizar simultáneamente el estado de múltiples laboratorios.

En esta línea, se ha desarrollado una interfaz gráfica que nos permite la conexión simultánea a dos laboratorios remotos y ver en todo momento las señales de salida y los parámetros de configuración de ambos. Como ejemplos de laboratorio se han utilizado los dos laboratorios remotos usados en los apartados anteriores, es decir, el laboratorio virtual del control del tanque y el laboratorio real de identificación de la respuesta temporal de un sistema.

La interfaz de laboratorio se ha creado como siempre, utilizando el *Plugin* desarrollado para EjsS y este caso se han añadido al modelo dos elementos de tipo *"Remote Lab"* que se configuran con las direcciones de los laboratorios monitorizados. En este caso, como el objetivo no es realizar experiencias sobre ninguno de ellos sino visualizar sus correspondientes salidas, no se utilizará ningún elemento visual de tipo *LabControl*. En el caso de que el objetivo hubiera sido trabajar con ambos laboratorios, se hubieran necesitado agregar dos elementos *LabControl*, uno para cada laboratorio a gestionar.

El elemento fundamental en esta interfaz es el elemento *LabLogin* que se ha configurado para que se conecte a los dos laboratorios añadidos al modelo. Esto ha sido posible debido a que ambos laboratorios disponen del mismo usuario con la misma contraseña que se utilizará para probarlo. Además, se han añadido dos gráficas que muestran en paralelo, los datos recibidos de cada laboratorio.

Con respecto al uso de esta página, conviene destacar que como ésta no pertenece a ningún laboratorio concreto, será una página que el profesor almacenará y utilizará en su equipo local, sin desplegar, por lo que el elemento *LabLogin* estará visible en todo momento. En cualquier caso, hacer esta página disponible remotamente desde una aplicación Node.js no debería resultar una tarea excesivamente complicada. La interfaz gráfica resultante ha sido la mostrada en la figura 45.

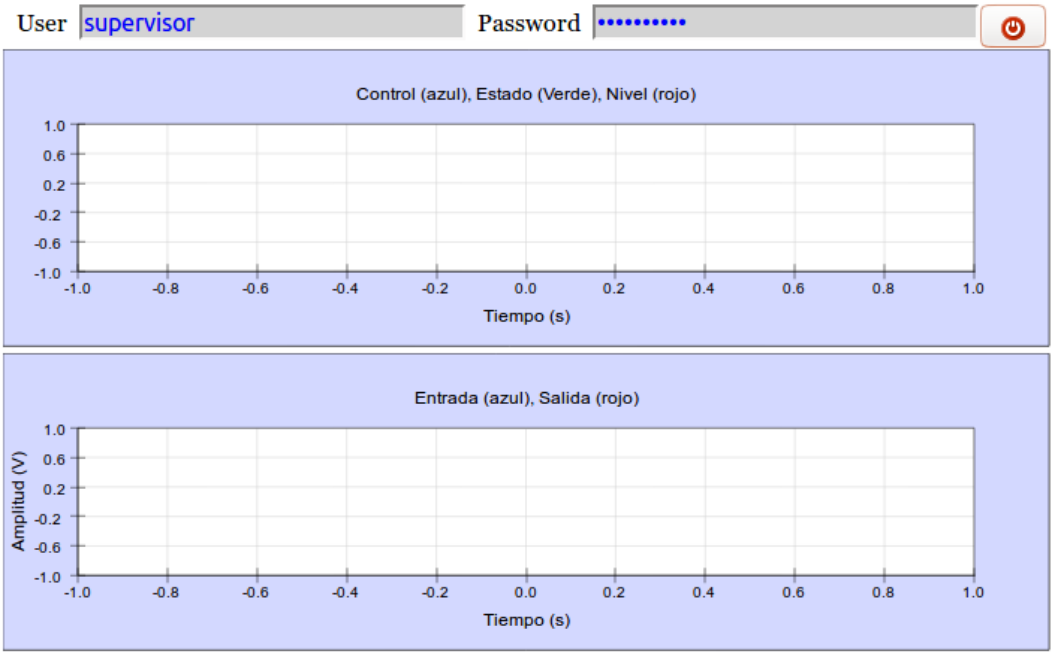


*Fig. 45. Supervisión de Laboratorio Virtual y Real Inactivos*

Para probar esta página, la hemos cargado en un explorador de internet y nos hemos conectado a los laboratorios utilizando un usuario supervisor. Inicialmente, los laboratorios estaban apagados por lo que las gráficas no mostraban actividad alguna tal y como se muestra en la figura 45.

Posteriormente, se procedió a iniciar el laboratorio virtual. Para ello, nos conectamos al mismo utilizando el usuario de un alumno autorizado y se pudo comprobar que la gráfica correspondiente a este laboratorio comenzaba a mostrar las salidas del mismo tal y como se muestra en la figura 46.

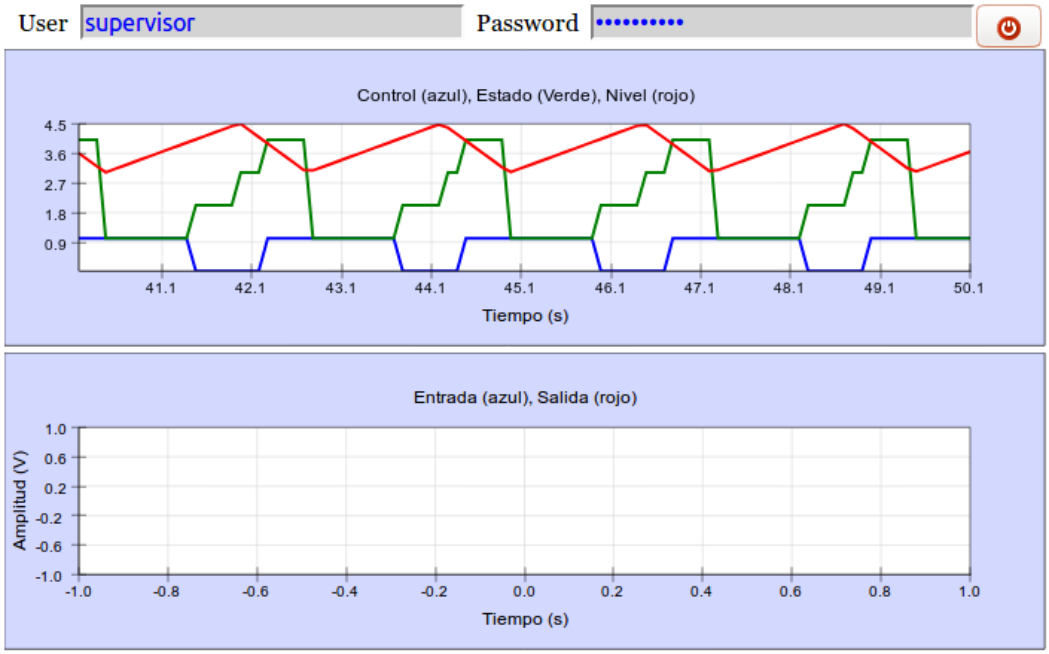


*Fig. 46. Supervisión de Laboratorio Virtual Activo*

Tras arrancar el laboratorio virtual, se procedió a arrancar el laboratorio real con las credenciales de otro usuario y automáticamente, la gráfica correspondiente a este laboratorio comenzó a mostrar sus salidas. La figura 47 muestra estos resultados.

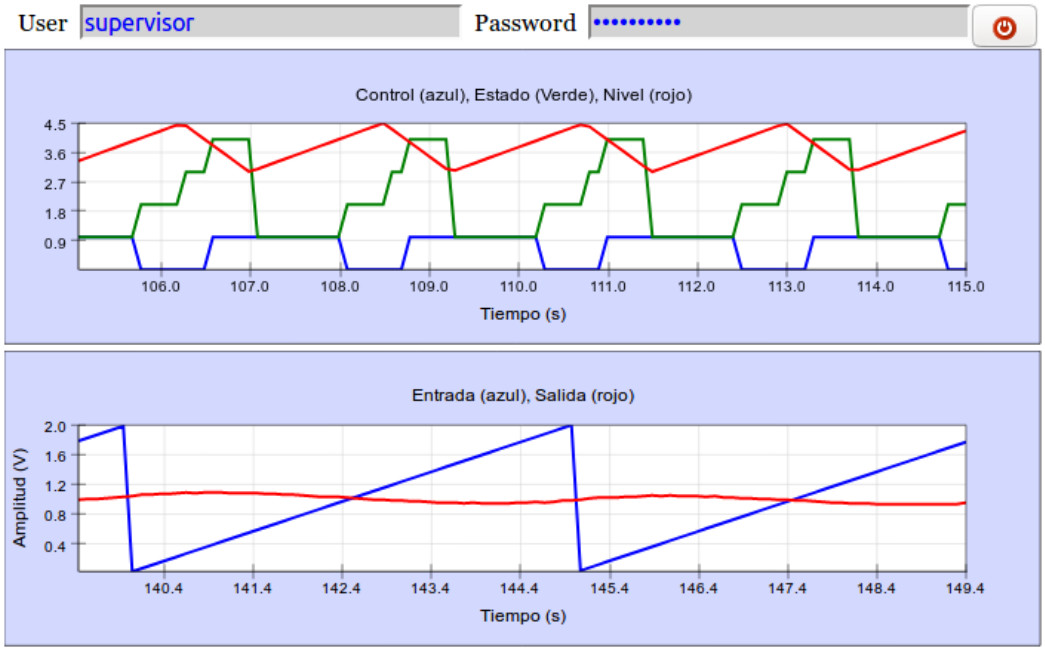


*Fig. 47. Supervisión de Laboratorio Virtual y Real Activos*

Una vez arrancados ambos laboratorios, se modificaron los parámetros de entrada de ambos y se comprobó que los cambios también eran visibles en las gráficas de la herramienta de supervisión desarrollada, tal y como se muestra en la figura 48.

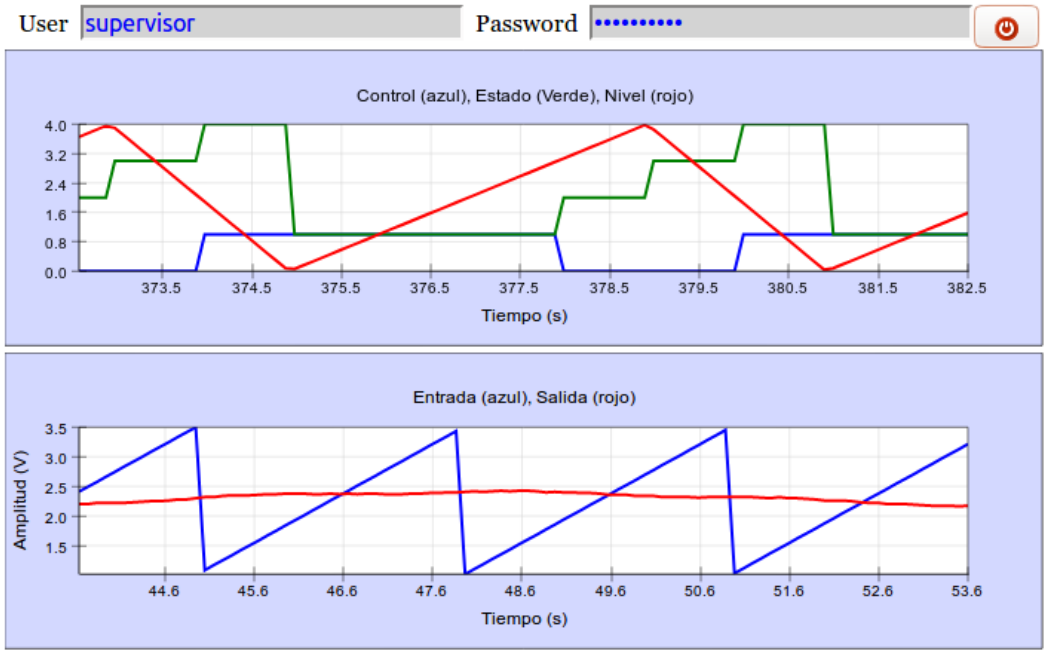


*Fig. 48. Supervisión de Laboratorio Virtual y Real Activos Modificados*

Finalmente, resaltar que actualmente, la aplicación Node.js permite monitorizar únicamente los parámetros de entrada en uso y las salidas del laboratorio remoto. No obstante, con pequeñas adaptaciones en la aplicación Node.js, el profesor podría llegar a supervisar diferentes aspectos de los laboratorios tales como el usuario conectado en cada momento o comprobar la conectividad de los laboratorios de los que es responsable.

# 6 Conclusiones y trabajos futuros

Para conseguir el objetivo principal del trabajo, la creación y despliegue de laboratorios remotos de forma cómoda, intuitiva y fácil para un usuario no avanzado utilizando EjsS, se han explorado diversas vías. La imposibilidad inicial de dar una solución elegante utilizando el sistema de extensiones existente en la herramienta ha llevado a implementar un nuevo sistema de extensiones basado en *Plugins* en el entorno EjsS, adicional al existente, que permita a diferentes desarrolladores ampliar y adaptar el entorno en base a sus necesidades. En nuestro caso, nos permite aportar una solución para la automatización del despliegue de laboratorios remotos y otra serie de ayudas relacionadas con éstos, todo ello de un modo totalmente integrado en una única herramienta.

Al concluir este trabajo y observar los resultados obtenidos se aprecia que se ha logrado cumplir todos los objetivos propuestos. Para ello, en la parte cliente se ha implementado el nuevo sistema de *Plugins* en EjsS sobre el que se ha creado un *Plugin* específicamente dedicado a los laboratorios remotos de la UCM y en la parte servidor, se han realizado diversas adaptaciones a la aplicación Node.js utilizada en el marco ReNoLabs. Todo ello, forma un sistema perfectamente válido y listo para su uso por parte del profesorado que así lo desée.

Desde el punto de vista de EjsS y los trabajos de extensión realizados, podemos realizar las siguientes afirmaciones:

- EjsS es una herramienta muy buena y completa no solo para las funciones de generación de simulaciones para la que fue diseñada, sino que ha demostrado también ser una herramienta perfectamente válida para la generación de interfaces HTML no destinados a ejecutar ninguna simulación.
- Los *Plugins* aportan una nueva dimensión y libertad para ampliar funcionalidad de modos totalmente personalizados y adaptados a las necesidades concretas de quien los desarrolla.
- El modo en el que se podría implementar la capacidad de extensión por *plugins* en EjsS es muy variada. La que se ha abordado en este trabajo tan solo es una de ellas por lo que se debe considerar tan solo como una propuesta que puede o puede no ser aceptada para ser incluida en el producto final.

Además desde el punto de vista de los laboratorios remotos, podemos decir que:

- El uso de EjsS para generar las interfaces gráficas de los laboratorios permite a su diseñador abstraerse de la complejidad de generar código JavaScript, permitiéndole focalizar todo su esfuerzo en las singularidades del laboratorio.
- La gestión de laboratorios remotos no es una tarea sencilla, pero en este trabajo se ha podido avanzar en ciertos aspectos que pueden ser aprovechados como base para trabajos futuros ya que aún queda mucho margen de mejora.

Como se ha comentado con anterioridad, el conjunto de funcionalidades implementadas es un subconjunto representativo del trabajo que se puede llegar a desarrollar, por lo que han quedado en el tintero una serie de trabajos de diferente complejidad. En la categoría de facilidades sencillas se pueden citar varias ideas:

- La pestaña “Remote Lab” implementada como extensión de modelo dispone de una zona libre en la que podrían mostrarse datos varios sobre el laboratorio existente en la dirección

configurada. Si la aplicación Node.js estuviera arrancada, EjsS podría descargar por ejemplo información sobre las entradas y las salidas disponibles, y mostrárselas al usuario.

- El uso de los laboratorios remotos está restringido a un conjunto de usuarios válidos que deben autenticarse antes de poder utilizarlos. La gestión de la lista de alumnos se realiza de modo manual en el servidor pero podría añadirse una extensión a EjsS que permitiera de manera cómoda añadir, editar y eliminar usuarios y gestionar sus permisos.
- Las extensiones implementadas permiten desplegar tanto el código del laboratorio generado por EjsS como un controlador de laboratorio implementado en C. Esto permite tener disponible el código del controlador que se ejecuta en cada laboratorio. Sin embargo, aún existe el riesgo de perder accidentalmente el archivo ".ejss" que contiene el código fuente del propio laboratorio si no se gestiona con cuidado. Por ello, resultaría más que interesante si a la vez que se despliega el laboratorio, se almacenara una copia del archivo EjsS que genera la interfaz desplegada.
- Igualmente, resultaría de interés implementar la descarga del archivo EjsS del servidor de modo que semejante al que se descarga el del controlador, la obtención de este archivo resulte cómoda para el profesor.

Estas funcionalidades añadirían un valor más o menos modesto al entorno con un esfuerzo relativamente bajo. En contraposición, existen una serie de funcionalidades cuyo esfuerzo de desarrollo sería bastante alto y que aportarían gran valor tanto a EjsS como al entorno de laboratorios. Entre ellas destacan:

- Los laboratorios remotos basados en la versión de ReNoLabs tomada para este trabajo únicamente permite a un alumno conectarse a la página de gestión del laboratorio. Con respecto a este punto, se están desarrollando soluciones para permitir un modo colaborativo entre alumnos, y eliminar esta restricción y explorar diferentes vías de uso de los laboratorios. El trabajo realizado y aquí presentado podría integrarse en la línea de desarrollo del actual ReNoLabs.
- La creación de nuevos elementos visuales para extender los existentes en EjsS no es una tarea sencilla y requiere de unos conocimientos relativamente avanzados de JavaScript y de la propia librería de controles que aporta EjsS. Tal vez se podría implementar una herramienta que facilite la creación de nuevos elementos o librerías de controles visuales para su uso en EjsS.
- El envío y compilación del código del controlador de laboratorio se ha implementado para aquellos laboratorios remotos basados en controladores C con compilador GNU. Esta funcionalidad se podría ampliar para incluir nuevos lenguajes de programación y generalizarlo para otros tipos de laboratorio.

Las ideas aquí aportadas se centran en la ampliación de diferentes áreas funcionales. Sin embargo, existe un aspecto muy importante a tratar relativo al despliegue de código fuente en el servidor. A modo demostrativo se ha desarrollado la funcionalidad de descarga y despliegue del controlador de laboratorio, que aporta muchas ventajas tanto a profesores como a alumnos en el ámbito educativo. No obstante, hay que indicar que esta característica también aporta un riesgo muy grande al entorno de laboratorio si el uso de la misma no se hace con un grado de cuidado o si se utiliza de un modo malintencionado. Hay que recordar que precisamente lo que se está transfiriendo al servidor es código fuente que el procesador remoto compilará y ejecutará sin ningún tipo de protección. Un

usuario malintencionado con el suficiente conocimiento del entorno y de las librerías disponibles en el servidor podría realizar acciones destructivas o poco lícitas en el sistema.

Por ello, se pone de manifiesto que esta funcionalidad requiere de la adopción de una serie de medidas de seguridad que garanticen la integridad del sistema y que lo proteja ante posibles ataques. El campo del estudio de la seguridad es un aspecto muy amplio y cuya implementación podría constituir en sí mismo material suficiente para otro Trabajo de Fin de Máster. En cuanto a la funcionalidad, considerada tal cuál está en estos momentos, únicamente debería ser utilizada por el profesorado responsable del laboratorio, aunque por supuesto puede hacerla extensible al alumnado una vez estudiados los riesgos que encierra su uso.

Para finalizar, como ya se ha comentado, el desarrollo de la infraestructura de *Plugins* de EjsS surgió en la reunión que se mantuvo junto con Francisco Esquembre en diciembre de 2017. Por supuesto, todo el desarrollo que se ha realizado para añadir este infraestructura en EjsS le será presentado a modo de propuesta para ser incluido en en el código fuente de EjsS. De ser aceptado, podría comenzar a formar parte de EjsS en futuras versiones de EjsS.

# Referencias

[1] "Laboratorios remotos: tecnología de la UNED de referencia internacional" http://portal.uned.es/portal/page?_pageid=93,53608911&_dad=portal&_schema=PORTAL

[2] "Remote Laboratory of Automatic Control", Universidad de León

http://lra.unileon.es/content/physicalsystems

[3] K. H. Johansson, "The quadruple-tank process: a multivariable laboratory process with an adjustable zero," in IEEE Transactions on Control Systems Technology, vol. 8, no. 3, pp. 456-465, May 2000.

[4] "Labs at Deusto", Universidad de Deusto

https://weblab.deusto.es/website/labs.html

[5] "Laboratorio de Automática", Universidad de Chile

http://ingenieria.uchile.cl/investigacion/presentacion/laboratorios/departamento-de-ingenieria-electrica/90676/laboratorio-de-automatica

[6] "University Network of Interactive Labs" (UNILabs), UNED

https://unilabs.dia.uned.es/blog/index.php?entryid=3

[7] "ReNoLabs", Julian Bermudez, https://bitbucket.org/juliberm/renolabs/src/master/

[8] E. Besada-Portas, J. Bermúdez-Ortega, L. de la Torre, J.A. López-Orozco, J.M. de la Cruz, "Lightweight Node.js & EJsS-based Web Server for Remote Control Laboratories", IFAC-PapersOnLine, Volume 49, Issue 6, 2016, Pages 127-132.

[9] J. Bermudez-Ortega, E. Besada-Portas, J. A. Lopez-Orozco, J. Chacon and J. M. de la Cruz, "Developing web & TwinCAT PLC-based remote Control laboratories for modern web-browsers or mobile devices," *2016 IEEE Conference on Control Applications (CCA)*, Buenos Aires, 2016, pp. 810-815.

[10] "EJS Home Page: Welcome to the wiki pages for Easy Java/Javascript Simulations!", Francisco Esquembre

www.um.es/fem/EjsWiki

[11] "Node.js", https://nodejs.org

[12] "Passport: Simple, unobtrusive authentication for Node.js", http://www.passportjs.org/

[13] "Embedded JavaScript templating", http://ejs.co/

[14] "Express: Fast, unopinionated, minimalist web framework for Node.js"

https://expressjs.com/

# Anexo A Infraestructura de *Plugins* en EjsS

El presente anexo se incluye para aportar una mejor visión de lo que son los *Plugins*, el modo en el que se instalan en EjsS e incluso se enumeran brevemente los elementos necesarios y pasos a seguir a la hora de implementar uno nuevo.

Una persona familiarizada con EjsS conoce la estructura de directorios que utiliza la herramienta. Como se muestra en la figura 49, los directorios principales que cuelgan del directorio de instalación son tres: *bin*, *doc* y *workspace*. Las carpetas *bin* y *doc* son propias de la aplicación y contienen los archivos binarios necesarios para ejecutar EjsS y diversa documentación. El contenido de estas carpetas suele ser fijo y no se suele modificar a no ser que se deseen modificar de algún modo las extensiones Java o JavaScript que de por sí aporta EjsS.

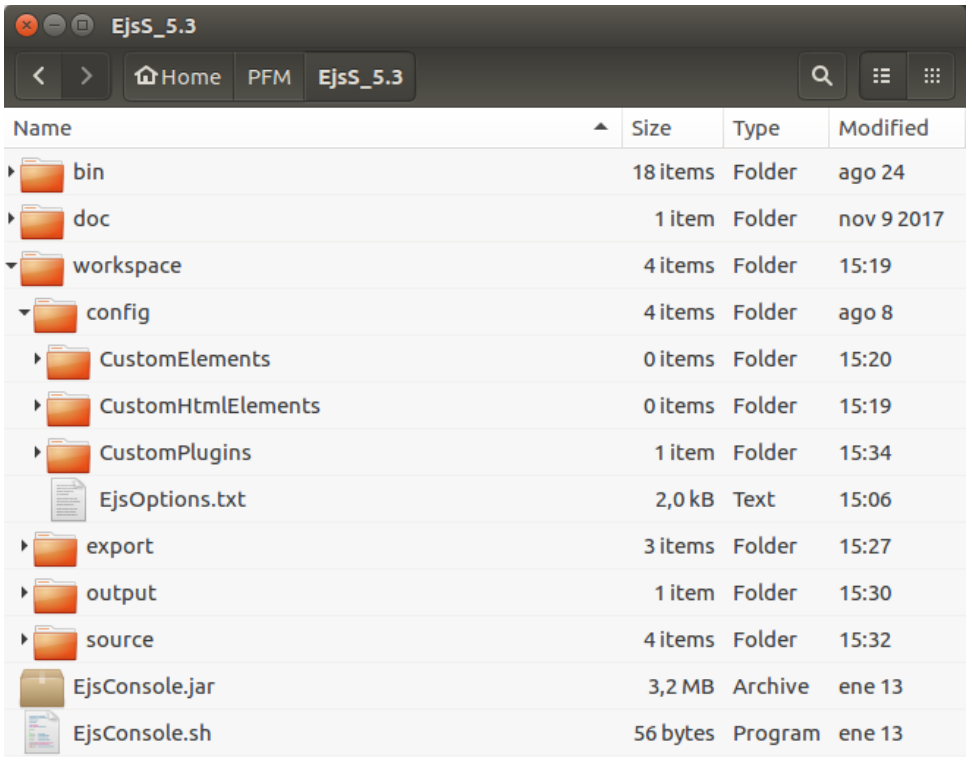


*Fig. 49. Árbol de Directorios de EjsS*

El directorio *workspace* está compuesto a su vez por otros 4 directorios destinados a almacenar el código fuente de las simulaciones que creamos, las exportaciones de las mismas e incluso las opciones de configuración con las que se arranca EjsS. Esta configuración se encuentra en el directorio *config* de *workspace* que a su vez contiene por defecto los directorios *CustomElements* y *CustomHtmlElements* en los que se almacenan los elementos visuales personalizados para simulaciones para Java y para JavaScript respectivamente.

Es en este directorio de configuración donde se encuentra la primera evidencia de la nueva infraestructura de *Plugins*. Como se aprecia de nuevo en la figura 49, en el directorio *config* se ha añadido un tercer directorio llamado *CustomPlugins*. Será en este directorio en donde el usuario deberá copiar los archivos *jar* que contienen los *Plugins*. Al igual que ocurre con las otras extensiones, EjsS explorará este directorio en busca de archivos que contengan *Plugins* y los instalará en el entorno en tiempo de ejecución. Para añadir un nuevo *Plugin* o actualizar uno

existente, bastará con copiar el nuevo archivo en esta carpeta y EjsS lo utilizará en el siguiente arranque. Igualmente, si se desea desactivar o eliminar un *Plugin*, bastará con eliminar el archivo correspondiente de esta carpeta y no quedará rastro de ellos en la siguiente ejecución de EjsS.

En nuestro caso, se han implementado todas las extensiones de EjsS en un único *Plugin* en un archivo llamado *ReNoLabs.jar* y éste es el único archivo que se necesitará para gestionar nuestros laboratorios remotos.

Los *Plugins* deben estar contenidos obligatoriamente en archivos *jar* que contienen las clases Java que EjsS instanciará. Para EjsS, los *Plugins* son todas aquellas clases que implementan la interfaz *org.colos.ejs.osejs.plugins.Plugin*. A continuación se describen muy brevemente los elementos necesarios a seguir para desarrollar un *Plugin*.

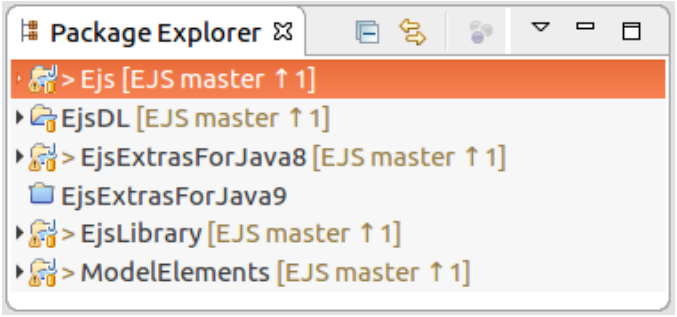


*Fig. 50. Estructura de Proyectos de EjsS en Eclipse*

Para desarrollar un *Plugin* se necesitará disponer bien de los archivos binarios de EjsS o bien su código fuente. Se recomienda partir del código fuente ya que de ese modo, se puede consultar la implementación de las clases y nos permite saber cuáles son las clases que podemos reutilizar y conocer mejor el funcionamiento interno de EjsS.

Los *Plugins* se deben implementan en Java, por lo que se necesitará un entorno de desarrollo para este lenguaje. En nuestro caso, el entorno utilizado es Eclipse, entorno muy extendido entre la comunidad desarrolladora de aplicaciones en Java, aunque se puede utilizar el que el entorno en el que el desarrollador se encuentre más cómodo. La figura 50 muestra la estructura de proyectos de EjsS una vez importado el código fuente en Eclipse.

Una vez se dispone de estos elementos, para crear un Plugin bastará con crear un nuevo proyecto Java y comenzar a desarrollar la clase que implementa la interfaz *Plugin* y las clases auxiliares que se necesiten.

Esta interfaz es la que permite a un desarrollador agregar a EjsS todos los elementos de expansión con los que se ha diseñado la infraestructura de *Plugins* que son los botones de la barra principal, las opciones principales, las opciones de modelo y los grupos de controles HTML descritos en el capítulo 4.

# Anexo B Actualizaciones de EjsS

La aplicación EjsS es una aplicación viva de la cuál se publican actualizaciones y mejoras con relativa frecuencia. Hay que tener en cuenta que una parte importante de este trabajo ha sido precisamente adaptar esta aplicación para poder añadir nuevas funcionalidades independientes al avance del desarrollo de la misma, pero este código no ha sido incorporado a la línea oficial de desarrollo de EjsS. Este trabajo tomó como base una versión intermedia de finales de 2017 en la línea de desarrollo de la versión 5.3 y ésta versión modificada es a día de hoy la que se debe instalar para poder realizar los desarrollos de laboratorios remotos descritos en este trabajo. Esta versión se incluirá como parte de los elementos que se entregarán junto a este trabajo.

Esta nueva infraestructura de *Plugins* es la propuesta que se realizará a Francisco Esquembre para su estudio. Si finalmente se acepta y se incluye este mecanismo dentro de la línea de desarrollo de EjsS, el usuario podrá elegir actualizarse a la última versión. En este caso, se deberá tener especial cuidado en comprobar previa instalación, que el interfaz expuesto por la nueva versión es compatible con la versión de los *Plugins* que se deseen utilizar. En el caso de que necesitar utilizar una versión superior de EjsS cuya interfaz no sea compatible con los *Plugins*, éstos deberán ser adaptados.

A la hora de tomar la decisión de actualizar EjsS a una versión superior, una de las cosas que hay que tener muy en cuenta es que la propia librería de elementos HTML que utiliza ha podido sufrir modificaciones que pueden suponer serios problemas de compatibilidad en simulaciones e interfaces de laboratorio previamente creadas. Como es obvio, el conjunto de elementos HTML que contiene está programado para funcionar con la versión EjsS con la que se distribuye y en ella podrían haberse creado nuevos elementos, podrían haberse eliminado elementos obsoletos, podrían haberse renombrado elementos existentes o incluso podrían haberse modificado las propiedades internas de elementos existentes. Estas modificaciones se ver reflejadas en EjsS en el conjunto de elementos disponibles en la vista HTML o en el conjunto de propiedades que se muestran al hacer doble clic sobre los elementos añadidos a la vista.

El archivo que contiene una simulación (archivo con extensión *ejss*), almacena entre muchas cosas, una lista exhaustiva de todos y cada uno de los elementos HTML, sus tipos y sus propiedades. Una simulación creada con una versión anterior de EjsS y que contenga algún elemento que se encuentre en alguna de las situaciones descritas, provocará que EjsS no pueda cargar el archivo correctamente al ser incapaz de emparejar la información contenida en el mismo con la información de la librería de elementos nueva. Esto resultará en errores cuya solución puede llegar a ser complicada de encontrar y podrían llegar a pasar por el poco aconsejable y arriesgado método de la edición manual del archivo *ejss*, método al que se ha tenido que llegar en alguna ocasión durante el desarrollo de este trabajo.

En el caso de la interfaz de laboratorio, hay que tener en cuenta que ésta también se encuentra definida dentro del fichero *ejss*, por lo que el problema de la actualización también las afecta. Los laboratorios remotos muestran las interfaces de los elementos tal y como están definidos en su librería de elementos. Esta librería no es otra que *ejsS.v1.min.js*. Esta librería también es la base de la librería de elementos HTML extendidos por ReNoLabs. Al crear un nuevo laboratorio, *ejsS.v1.min.js* se copia en una carpeta del servidor por lo que mientras no se modifique, ambas librerías serán coherentes entre sí y se comportarán correctamente. En el caso de utilizar una versión de EjsS diferente para crear una nueva interfaz, corremos el riesgo de utilizar un elemento incompatible bien con la librería *ejsS.v1.min.js* o bien con la librería de elementos de ReNoLabs.

Esta incompatibilidad la encontraremos en el mejor de los casos en el momento en el que se realicen las pruebas locales (caso de incompatibilidad con ReNoLabs) o en el peor de los casos tras haber desplegado la interfaz al laboratorio (caso de incompatibilidad con *ejsS.v1.min.js*). En este último caso, corremos el riesgo de dejar sin servicio un laboratorio.

¿Significa todo esto que no se puede hacer uso de las nuevas versiones de EjsS con la extensión de laboratorios remotos y que nos tenemos que quedar anclados en la versión utilizada durante el desarrollo? En absoluto. Lo que se pretende en en este punto no es decretar que no se pueda actualizar EjsS ni que se desaconseje hacerlo, sino hacer hincapié en la necesidad de estudiar la viabilidad de las actualizaciones y sobre todo aportar conciencia de los posibles efectos secundarios.