Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

Tipo de artículo: Artículo original
Temática: Soluciones informáticas
Recibido: 10/06/2018 | Aceptado: 20/09/2018 | Publicado: 28/09/2018

Plataforma para la integración de componentes en un Sistema de Laboratorio Remoto

Platform for the integration of components in a Remote Laboratory System

Eric Gutiérrez Cabrera [1*], Yisell Gato Alvarez [2]

[1] Facultad de Ciencias y Tecnologías Computacionales. Universidad de las Ciencias Informáticas. erickd@uci.cu
[2] Facultad de Ciencias y Tecnologías Computacionales. Universidad de las Ciencias Informáticas. ygato@uci.cu

[*] Autor para correspondencia: ygato@uci.cu

Resumen

El surgimiento de las Tecnologías de la Información y las Comunicaciones (TICs) y la evolución de la World Wide Web (WWW) han sido factores de vital importancia en la transformación de la sociedad, especialmente en el área educacional. El desarrollo de la educación a distancia ha permitido a docentes y especialistas el intercambio de información y de recursos tecnológicos a través de las redes de comunicación sin que sea necesario el contacto físico; recursos que por problemas económicos no se tienen al alcance de todos los centros investigativos y educacionales del país y que a través de la Web se pueden utilizar desde cualquier lugar y momento que se estime conveniente. La presente investigación se enmarca en la concepción e implementación de una plataforma que permita integrar componentes de un Sistema de Laboratorios Remoto (SLR) en el área de automática con el objetivo de facilitarle a quien lo necesite las herramientas necesarias para el desarrollo de tareas investigativas que se llevan a cabo en muchas instituciones y que las mismas no constan de la instrumentación necesaria para desarrollarlas. El proceso de desarrollo del software estuvo guiado por la metodología mínima, completa y extensible: Open Up y se utilizó como lenguaje de programación Java con el fin de obtener buenos resultados.

Palabras clave: Laboratorios Remotos; plataforma; aprendizaje electrónico.

Abstract

The emergence of Information and Communication Technologies (ICTs) and the evolution of the World Wide Web (WWW) have been vital factors in the transformation of society, especially in the educational area. The development of distance education has allowed teachers and specialists to exchange information and technological resources through communication networks without physical contact being necessary; resources that due to economic problems are not available to all research and educational centers in the country and that through the Web can be used from

any place and time that is deemed appropriate. The present investigation is part of the conception and implementation of a platform that allows integrating components of a Remote Laboratory System (SLR) in the area of automatic with the aim of providing the necessary tools for the development of investigative tasks to those who need it. they are carried out in many institutions and that they do not have the necessary instrumentation to develop them. The software development process was guided by the minimum, complete and extensible methodology: Open Up and it was used as a Java programming language in order to obtain good results.

Keywords: *Remote Laboratories; platform; electronic learning*

Introducción

Desde el comienzo mismo de la evolución del hombre la información ha jugado un papel fundamental en la vida de este y ha progresado paralela y conjuntamente hasta convertirse en parte indispensable e indiscutible de la práctica diaria a todos los niveles(DUMÉNIGO 2012). La gran cantidad de información manejada, genera un haz de conocimientos que aumenta de forma acelerada, lo cual trae consigo una mayor demanda de productos de *software*, soluciones informáticas y servicios que permitan gestionarla eficientemente. Por estas razones fue obligatoria la interrelación de esta materia con otras ciencias y aspectos de la vida social. Así surgen las Tecnologías de la Información y las Comunicaciones (TICs), las cuales han llegado a ser uno de los pilares primordiales de la sociedad actual.

Las TICs son aplicables en casi todos los sectores de la sociedad reportando continuas transformaciones en el modelo económico, social y cultural e incidiendo en la mayoría de los aspectos de la vida cotidiana alcanzando mejoras en la calidad de vida y mayor eficiencia en los procesos de gestión de la información. El conjunto de tecnologías que abarcan las TICs es variado, destacando las Tecnologías Web, las cuales sirven para acceder a los recursos de conocimiento disponibles en la red de redes, más conocida como Internet (KO *et al.* 2019), (MAR and GULÍN 2018), (MARTÍN-PENA *et al.* 2018).

Cuba es un país que debido a su situación económica le es difícil poner a disposición de cada centro educativo todos los recursos tecnológicos necesarios para el aprendizaje de los estudiantes. El desarrollo de un Sistema de Laboratorios Remoto (SLR) sería una buena opción para la solución de este problema. A través de las redes de comunicaciones se realizarían intercambios de recursos entre los estudiantes y los laboratorios a la hora de realizar las prácticas; contribuyendo así a la reducción de importaciones de materiales técnicos para la confección de Laboratorios Convencionales (LC) y se les facilitaría a los alumnos el aprendizaje del contenido de las materias al permitirles estudiar en el momento y lugar que crean conveniente.

En el año 2002 fue creada la Universidad de las Ciencias Informáticas (UCI) con el objetivo de formar profesionales capacitados en el desarrollo de *software*. La UCI es un centro que posee amplios recursos tecnológicos y computacionales para el procesamiento de grandes cálculos y desarrollo de aplicaciones. Además, posibilita la descarga de programas y artículos científicos dado al factible acceso a la red de redes, Internet (BARQUET *et al.* 2018),(MAR *et al.* 2017).

En la facultad 6 perteneciente a dicha universidad se encuentran en desarrollo un conjunto de laboratorios virtuales enfocados al área de la automática. Estos permiten a los especialistas y estudiantes de esta área realizar prácticas basadas en simulaciones, observaciones y análisis de muestras. Sin embargo, la utilización de cada uno de ellos se realizaría de manera independiente haciendo engorroso el acceso simultáneo a más de un laboratorio a la hora de realizar las prácticas, provocando así una posible pérdida de tiempo e información. Para solucionar este problema sería más conveniente agruparlos todos en un mismo sistema que brinde estos servicios de forma integrada, segura y confiable. Planteándose como objetivo general de la investigación desarrollar una plataforma para la integración de componentes en el Sistema de Laboratorios Virtuales Remotos.

Materiales y métodos o Metodología computacional

En la presente sesión se describe la solución propuesta para obtener una noción en general de la Plataforma para la integración de componentes en el Sistema de Laboratorios Virtuales y a Distancia. Se identifican los requisitos funcionales y no funcionales del sistema, los casos de uso, patrones y se modelan los diagramas pertinentes para proceso de diseño logrando así guiar el proceso de desarrollo de software.

Especificación de los Requisitos del sistema.

El principal objetivo de determinar las funcionalidades que tendrá el sistema es guiar el desarrollo de la aplicación hacia la obtención de un diseño que cumpla con las expectativas requeridas por el cliente. La especificación de requisitos es la actividad que enumera y describe los requisitos necesarios para el sistema. Se dividen en dos grupos los funcionales (expresan capacidad) y los no funcionales (expresan cualidad).

Requisitos Funcionales del sistema: Los requisitos funcionales (RF) de una aplicación son las capacidades o condiciones que el sistema debe cumplir para satisfacer las necesidades primarias del cliente.

A continuación se definen los requisitos funcionales identificados para el desarrollo del sistema.

RF1: Autenticar usuario.

RF2: Adicionar usuario.

RF3: Editar usuario.

RF4: Eliminar usuario.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

RF5: Buscar usuario por nombre, segundo nombre, apellido, nombre de usuario, dirección de correo y estado.

RF6: Asignar permisos a los usuarios.

RF7: Adicionar *portlet*.

RF8: Activar *portlet*.

RF9: Desactivar *portlet*.

RF10: Eliminar *portlet*.

RF11: Mostrar porciento del rendimiento de la memoria RAM mediante una gráfica de pastel.

RF12: Mostrar porciento del rendimiento de la memoria de intercambio mediante una gráfica de pastel.

RF13: Mostrar porciento del rendimiento del CPU mediante una gráfica de hilos.

RF14: Mostrar porciento de las peticiones del portal mediante una gráfica de pastel.

RF15: Mostrar porciento de las peticiones de cada *portlet* mediante una gráfica de pastel.

Requisitos no Funcionales del sistema: Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener, aunque no formen parte de su función.

A continuación se definen los requisitos no funcionales identificados para el desarrollo del sistema.

Usabilidad: El sistema podrá ser utilizado por usuarios con conocimientos básicos en el área de la Automática contando con interfaces intuitivas para un mejor desenvolvimiento de los usuarios. Tendrá buena visibilidad en los principales navegadores como Mozilla Firefox, Opera y Chrome.

***Software*:** Debe estar instalado:

En los ordenadores de los usuarios:

- El navegador Web Mozilla Firefox versión 32.0 o superior.

En las estaciones que fungen como servidores:

- Servidor Web, Apache Tomcat 7.0.2 con la aplicación *Liferay* Portal 6.2.0.

Hardware: El servidor de BD debe tener más de 1GB de RAM y 80GB o más de capacidad de disco duro.

- El servidor de aplicaciones debe tener 2 GB de RAM o superior y 80GB o más de capacidad de disco duro. Un microprocesador igual o superior al Intel Dual Core, con una velocidad de procesamiento de 2.00 GHz en adelante.

Restricciones en el diseño y la implementación:

- Se utilizará el estándar de codificación propuesto por el lenguaje de programación Java.
- Para la programación en el lado del cliente se utilizará JavaScript con la librería Alloy UI para enriquecer la aplicación.
- Para las llamadas asíncronas al servidor se utilizará la tecnología AJAX con asistencia de la librería Alloy UI.

- Se utilizará la librería ATACH.DLL para la interconexión de las máquinas virtuales del apache Tomcat donde se encuentra JMXBridge y la máquina virtual de Liferay.

Interfaz: La interfaz de la plataforma estará desarrollada mediante la interfaz de usuario de Liferay Portal.
Modelo de Casos de Uso del Sistema: El modelo de casos de uso del sistema permite que los desarrolladores y los clientes lleguen a un acuerdo sobre los requisitos, es decir sobre las condiciones y posibilidades que debe cumplir el sistema. Los CU describen las interacciones entre uno o más usuarios y el sistema, con el fin de proporcionar un resultado observable de valor para el actor.
Los CU describen las interacciones entre uno o más usuarios y el sistema, con el fin de proporcionar un resultado observable de valor para el actor (FRAMEWORK 2011)
Actores de sistema: A continuación se enuncian y describen los actores que interactúan con el sistema.

Tabla 1: Descripción de los actores del sistema.

Actor del sistema	Descripción
Usuario	Representa una persona. Es la encargada de autenticarse en el sistema.
Administrador	Representa una persona. Es la encargada de gestionar los usuarios que acceden al sistema, administrar los *portlet* que se visualizan en el mismo y además supervisa el monitoreo del servidor de la plataforma.

Patrones de casos de uso del sistema: Los patrones de casos de uso son comportamientos que deben existir en el sistema, ayudan a describir qué es lo que el sistema debe hacer, es decir, describen el uso del sistema y cómo este interactúa con los usuarios. Estos patrones son utilizados generalmente como plantillas que describen como debería ser estructurados y organizados los casos de uso. Son patrones que capturan mejores prácticas para modelar casos de uso.
Los patrones de CU identificados fueron:

- Múltiples Actores: Este patrón se pone de manifiesto en el CU: Autenticar usuario ya que al mismo ingresan más de un actor.
- CRUD Total o Completo: Este patrón se evidencia en el CU: Gestionar usuario ya que permite modelar las diferentes operaciones para gestionar una entidad de información, tales como adicionar, modificar, buscar y eliminar.

Resultados y discusión

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

La siguiente figura (Fig. 1) evidencia el modelo de casos de uso de la plataforma para la integración de componentes en el SLr. Se evidencia en la misma los actores del sistema relacionados con los casos de uso que inicializan.

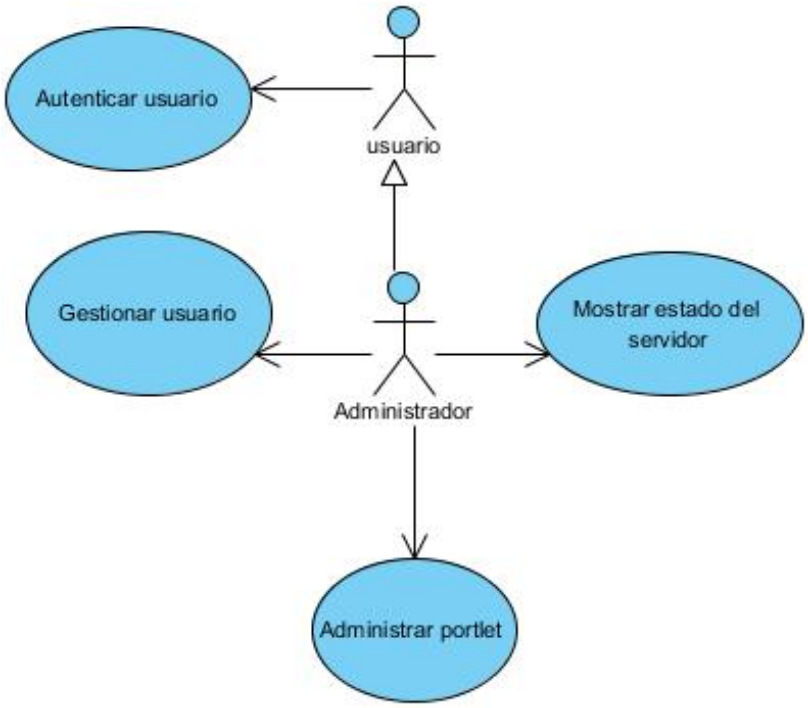

Fig.1: Diagrama de casos de uso del sistema.

Descripción de casos de uso del sistema.

La especificación de los casos de uso hace referencia a la descripción de cada una de las partes antes definidas para lograr un mayor entendimiento del funcionamiento de las mismas. Muestran detalladamente paso a paso cómo funciona el sistema ante una acción ejercida por el actor, en este caso, el administrador del sistema. A continuación se describirá el CU Mostrar estado del servidor, por ser el más significativo desde el punto de vista arquitectónico.

Tabla 2: CU: Mostrar estado del servidor.

Objetivo	Mostrar en qué estado se encuentran los componentes del servidor de la plataforma para tener un control del mismo.
Actores	Administrador del sistema (inicia)
Resumen	El caso de uso inicia cuando el actor necesita visualizar la información referente al monitoreo de la plataforma. Se muestran cinco gráficos en

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

	porciento con la información referente a los rendimientos de la memoria RAM, memoria de intercambio, CPU, peticiones al portal y peticiones a los *portlets* respectivamente. Finaliza así el CU.		
Complejidad	Alta		
Prioridad	Crítico		
Precondiciones	El usuario debe tener el rol de administrador del sistema.		
Postcondiciones	Se muestran los estados de las memorias, el CPU, peticiones al portal y peticiones a los *portlets* del sistema.		
Flujo de eventos			
Flujo básico Mostrar estado del servidor de la plataforma.			
	Actor	**Sistema**	
1.	El administrador del sistema selecciona la opción ´´Monitoreo del servidor´´		
2.		Muestra la información referente al estado de las memorias, CPU, peticiones al portal y peticiones a los *portlets* finalizándose así el caso de uso.	
3.			
Relaciones	**CU Incluidos**	No precede.	
	CU Extendidos	No procede.	
Requisitos no funcionales	No procede.		
Asuntos pendientes	No procede.		

Arquitectura del sistema: La arquitectura de *software* define la estructura de un sistema. Esta estructura se constituye de componentes, módulos o piezas de código que nacen de la noción de abstracción, cumpliendo funciones específicas e interactuando entre sí con un comportamiento definido. Los componentes se organizan de acuerdo a

ciertos criterios, que representan decisiones de diseño. En los últimos años la arquitectura de *software* se ha debatido en dos corrientes fundamentales: los estilos arquitectónicos y los patrones arquitectónicos.

Patrones arquitectónicos: Un patrón arquitectónico es la expresión de un esquema estructural de organización para sistemas de software, que proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye normas y directrices para las relaciones entre ellos(GARLAND and ANTHONY 2003).

Modelo Vista Controlador: El patrón MVC permite separar los componentes de aplicación en tres niveles, interfaz, lógica y modelo (acceso a datos). Es una especialización de un modelo de capas, con la diferencia que se usa para entornos Web como patrón por excelencia.

Entre las ventajas que este modelo de arquitectura presenta, es importante mencionar las siguientes:

- Existe una clara separación entre los componentes de un programa; lo cual nos permite implementarlos por separado.
- Se puede reemplazar los componentes del sistema (el Modelo, la Vista o el Controlador) sin mucha dificultad.(CATALANI 2007).

Las responsabilidades de las capas en MVC son:

Capa modelo: Representa los datos del programa, los maneja y controla todas sus transformaciones.

Capa controladora: Es el eje central de nuestra arquitectura, encargada de gestionar todas las peticiones, validar los inputs recibidos y dirigir cualquier petición de cualquier tipo.

Capa vista: Es la respuesta de cada controlador y lo que se le presenta al usuario final, se puede comunicar con el controlador y el modelo (en algunas ocasiones).

A continuación, en la figura 2 se muestra el funcionamiento del patrón de diseño MVC evidenciándose en él, las tres capas anteriormente mencionadas:

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

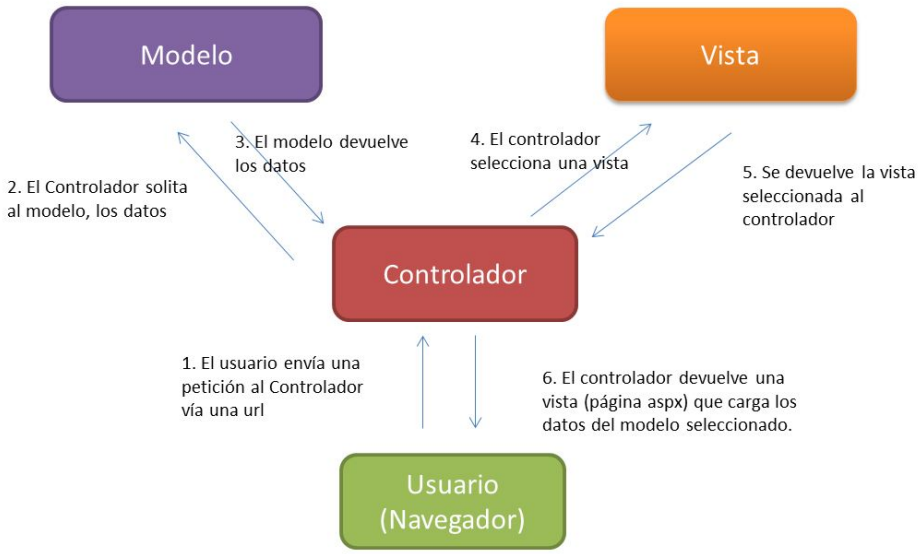

Fig.2: Funcionamiento del patrón Modelo Vista Controlador.

Cliente-Servidor: Desde el punto de vista funcional, se puede definir la computación Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma(MÁRQUEZ and ZULAICA 2004).

En el modelo cliente servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor, y este envía uno o varios mensajes con la respuesta (ver Fig.3).

Fig. 3: Modelo Cliente/Servidor(MÁRQUEZ and ZULAICA 2004).

La plataforma para la integración de componentes en el SLR presenta los patrones arquitectónicos MVC y Cliente-Servidor. Este último se manifiesta en el paradigma petición-respuesta, dado que es una aplicación Web a la que se accede por el protocolo HTTPS.

Diagramas de clases del diseño.

Los diagramas de clases muestran cómo se lleva a cabo la colaboración entre las clases para dar cumplimiento a un requisito determinado. Para la elaboración de estos diagramas se hace uso de estereotipos que ayudan a representar de manera más fácil la función y el carácter de las clases dentro de la realización del requisito.

En la figura 4 se muestra el diagrama de clases del diseño del CU Mostrar estado del servidor, donde se evidencia el patrón arquitectónico MVC ya que se desglosa el mismo en 3 paquetes, Vista, Controladora y Web Services o Modelo. En el paquete Vista se encuentra la interfaz que interactúa con el usuario, compuesta por un formulario con todos los *portlets* a monitorear. La misma envía peticiones a la clase **PortletStatistics** del paquete Controlador, estas peticiones se tramitan al JMXBridge el cual obtiene la información de los MBeansServer para luego reenviarla a la clase del paquete Controlador nuevamente para mostrar esa información mediante la interfaz del paquete Vista.

En la figura 5 se visualiza el diagrama de clases de la API JMXbridge del paquete Web Services. Este está compuesto por dos paquetes, Vista y Controlador, donde en el primero se tiene un cliente que hace un link con la clase controladora **DispatcherServlet** que a su vez redirecciona a la controladora **StatisticsController**. Esta usa las clases **PortletstatisticsImpl** y **PortalStatisticsImpl**, ambas utilizan **JMX_Utils** que a su vez usa la librería ATACH.DLL. Finalmente se devuelve la información formato JSOM.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

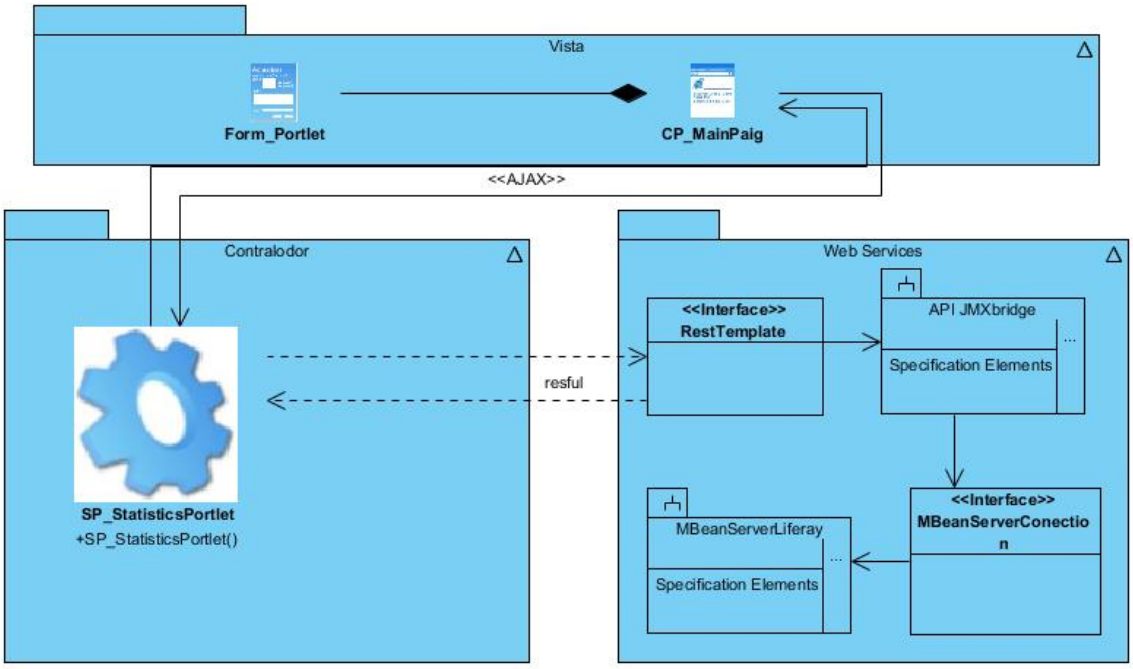

Fig.4: Diagrama de clases del diseño del CU Mostrar estado del servidor.

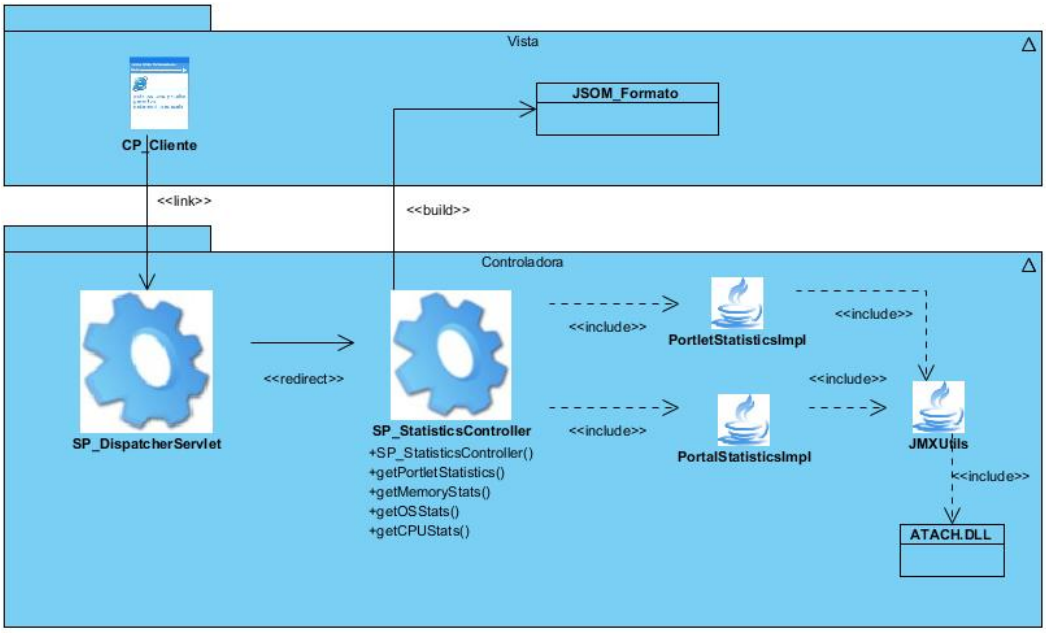

Fig.5: Diagrama de clases del diseño de JMX Bridge.

Patrones de diseño: Los patrones de diseño de software son el esqueleto de las soluciones a problemas comunes en el desarrollo de *software*. Brindan una solución a problemas de desarrollo de *software* que están sujetos a contextos

similares. Los patrones utilizados en la presente investigación fueron, GRASP, más conocidos como patrones de asignación de responsabilidades, los patrones GoF[1] y el Modelo Vista Controlador (MVC).

GRASP : Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones(LARMAN 1999). A continuación se mencionan los utilizados en el sistema.

- Patrón Experto: Se basa en asignar una responsabilidad a la clase que cuenta con la información necesaria para cumplir dicha responsabilidad. Permite conservar el encapsulamiento, ya que los objetos se valen de su propia información para realizar lo que se le oriente. Este patrón se evidencia en la clase **JMXUtils**, el cual inicializa el **MBeanServerConnection** y por ende es el encargado de verificar si el **MBeanServer** de *Liferay* Portal está en ejecución para establecer una conexión con el mismo.
- Patrón Creador: El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El objetivo fundamental de este patrón es encontrar un creador que se conecte con el objeto producido en cualquier evento. Al comportarse como creador, se da soporte al bajo acoplamiento. Teniendo en cuenta que se utilizó el marco de trabajo de Spring, se hace uso del IoC Container, del BeanFactory y Spring Contexto los cuales se encargan de instanciar las clases anotadas dentro del paquete "com.uci.*Liferay*.jmx.bridge", dicha configuración se encuentra en el archivo "servlet-context.xml" del **JMXBridge** como se muestra a continuación:

  <context:component-scan
  basepackage="com.uci.*Liferay*.jmx.bridge" />
  <mvc:annotation-driven />
- Patrón Bajo Acoplamiento: Este patrón asigna una responsabilidad para mantener el bajo acoplamiento. La idea es tratar de que una clase no dependa de muchas otras, así esa clase no tendrá muchas dependencias, lo que facilitará la reutilización de la misma y se reducirá el impacto de los cambios. Se evidencia este patrón manteniendo las relaciones mínimas entre clases.
- Patrón Alta Cohesión: Una clase con alta cohesión tiene un número relativamente pequeño de métodos, con funcionalidades altamente relacionadas, y no realiza mucho trabajo, colaborando con otros objetos para compartir el esfuerzo si la tarea es extensa. Las clases con alta cohesión son relativamente fáciles de mantener, entender y reutilizar. En el sistema se hace evidente la alta cohesión ya que las clases tienen una función bien definida dentro del sistema. Alta cohesión y bajo acoplamiento se retroalimentan juntos, gracias

[1] Conocidos como patrones de la pandilla de los cuatro (GoF, siglas en inglés de Gang of Four).

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

a que Java es un lenguaje Orientado a Objetos equilibramos las funcionalidades de tal forma que una clase no estuviera sobrecargada y que en caso de mantenimiento o errores estos fueran encontrados fácilmente. Por ejemplo, el controlador "**StatisticsController**" utiliza dos objetos para acceder a las estadísticas del portal y de los *portlets* respectivamente, estos a su vez dependen únicamente de un objeto **JMXUtils** para acceder a las propiedades de los **MBeans** publicados por *Liferay*. En la siguiente figura se muestra un pequeño fragmento del diagrama de clases del diseño del API JMXBridge donde se evidencia la clase en la que los patrones: alta cohesión y bajo acoplamiento se ponen de manifiesto.

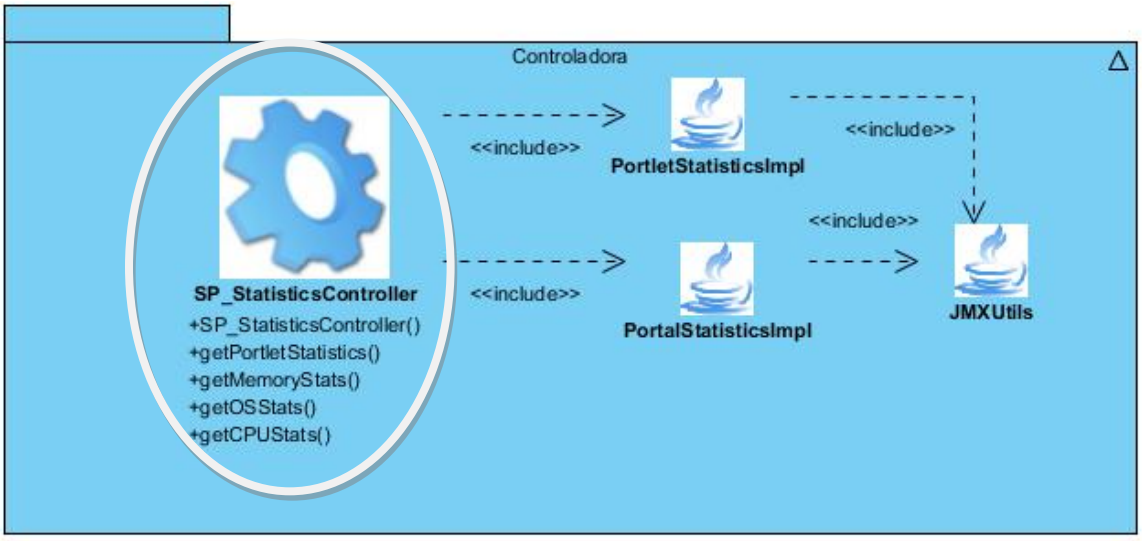

Fig.6: Patrones de diseño GRASP: Alta cohesión y bajo acoplamiento.

- Patrón Controlador: Para manejar y controlar los eventos del sistema. Con la utilización de este patrón se logra separar la lógica de negocio de la capa de presentación. De esta manera se logra un mayor control sobre el sistema y se favorece la reutilización de código. El sistema utiliza el **DispatcherServlet** de Spring como controlador frontal, una vez llegada una petición asigna el controlador "**StatistcisController**" para que la responda, en dependencia de la URL solicitada se retorna la información correspondiente en formato JSON. En el pequeño fragmento del diagrama de clases del diseño del API JMXBridge que a continuación se muestra, se evidencia lo anteriormente planteado, resaltando en dicho fragmento la clase donde el patrón Controlador se pone de manifiesto.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

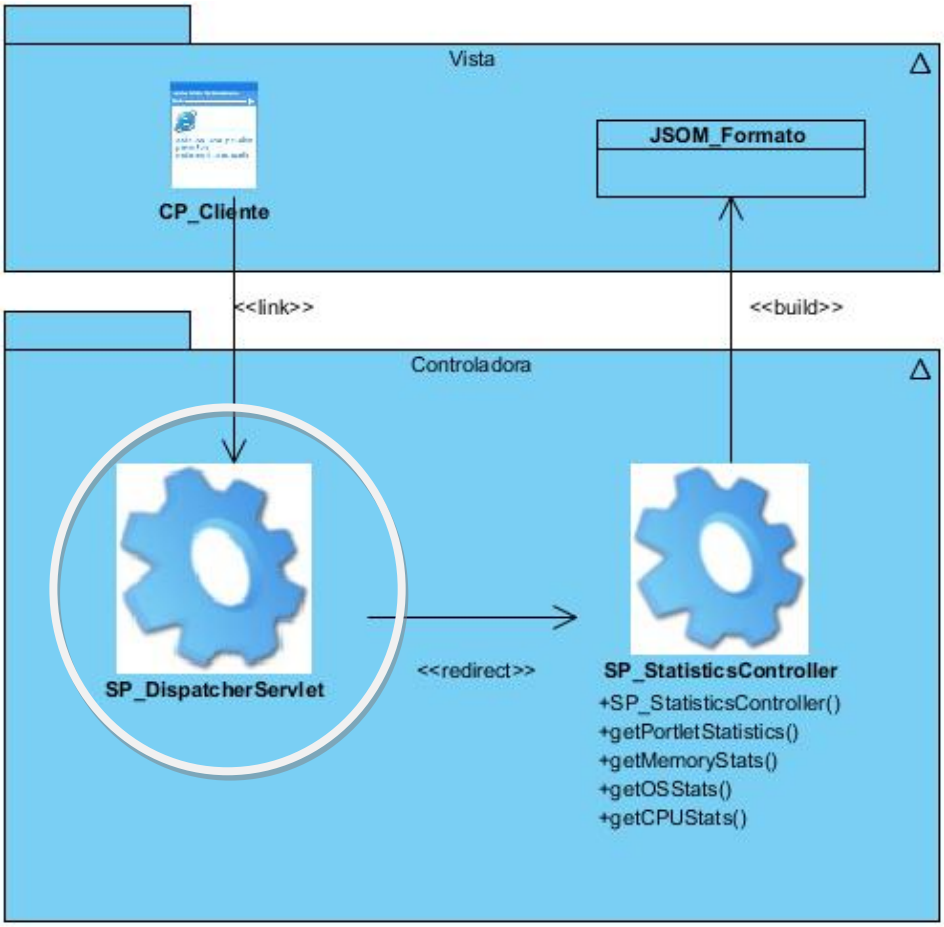

Fig.1: Patrón de diseño GRASP: Controlador.

GoF

Los patrones GoF complementan a los patrones GRASP y en ocasiones se puede encontrar una contraposición entre este tipo de patrones, e incluso, podría inferirse que algunos patrones GoF son variantes de los patrones GRASP, es por ello que la decisión de utilizar uno u otro debe tomarse con precaución y aplicarse sólo en el ámbito necesario. Entre los patrones GoF más utilizados se pueden citar el de Agente, Fachada y Agente Dispositivo y el Comando.

- El patrón Agente explica el comportamiento cuando no se desea o no es posible acceder directamente a un componente, para ello este patrón sugiere definir una clase sustituta de *software* que represente al componente y asignarle la responsabilidad de comunicarse con el componente real. Debido que la clase **StatisticsPortlet** no puede obtener directamente del **MBeanServerLiferay** la información referente al monitoreo del sistema, se define la API JMXBridge como intermediaria entre ambas clases, asumiendo la responsabilidad de acceder

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 11, No. 9, Mes Septiembre, 2018
ISSN: 2306-2495 | RNPS: 2343
Pág. 33-49
http://publicaciones.uci.cu

al **MBeanServerLiferay** y obteniendo de esta la información solicitada por la **StatisticsPortlet**. La siguiente figura es un fragmento del diagrama de clases del diseño del CU: Mostrar estado del servidor, la misma se evidencia lo anteriormente planteado.

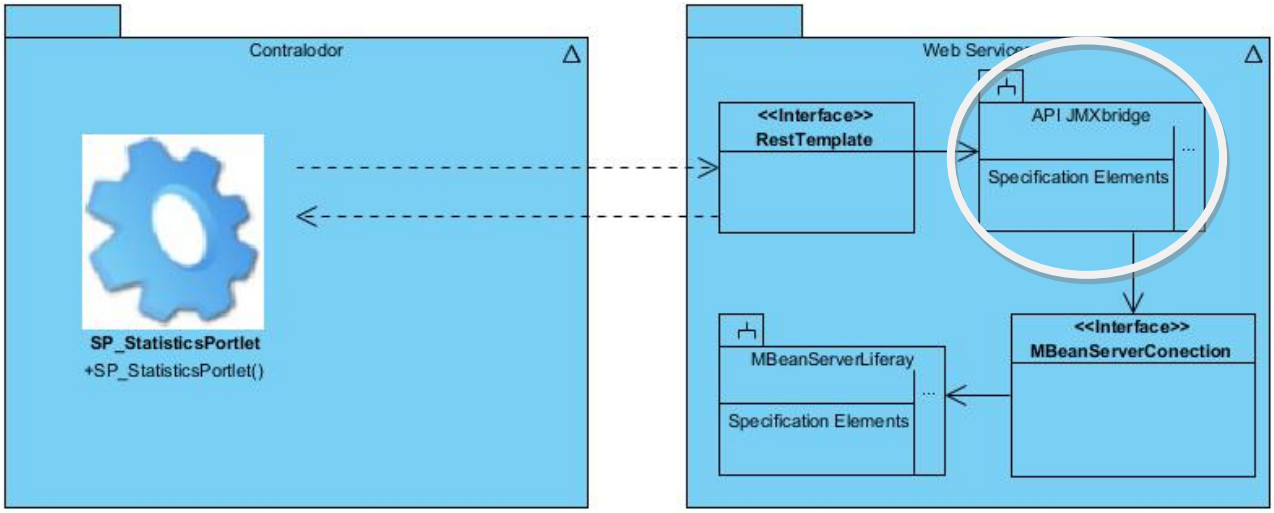

Fig.2: Patrón de diseño GoF: Agente.

- El patrón Fachada es generalmente utilizado para resolver problemas de integración, y se aplica cuando se requiere una interfaz común de comunicación con un conjunto de interfaces o funciones de otro subsistema, en este sentido Fachada define una sola clase que unifique las interfaces y le asigna la responsabilidad de colaborar con el subsistema. El patrón Agente Dispositivo es una especificidad de Fachada cuando el subsistema que se quiere integrar es un dispositivo externo.
- El patrón Comando soluciona el problema de cuando un objeto o sistema recibe varias peticiones o comandos, para ello cada comando define una clase que lo represente y le asigna la responsabilidad de ejecutarse el mismo, de esta forma reduce la responsabilidad del receptor en el manejo de los comandos, aumenta la facilidad con que pueden agregarse otros comandos y ofrece las bases para registrar los comandos, formar colas de espera con ellos y cancelarlos.

Conclusiones

A partir del análisis y diseño de la solución propuesta, para la integración de componentes en el SLR permitió una mejor comprensión del problema y fueron además definidos los requisitos funcionales y no funcionales de la misma.

Los RF identificados se agruparon en el modelo de casos de uso del sistema, destacando por su impacto en la arquitectura el CU: "Mostrar estado del servidor", el cual se describe dividido en secciones con el objetivo de lograr una mayor comprensión del mismo.

Se definieron los patrones arquitectónicos y patrones de diseño que posibilitaron definir una mejor arquitectura, entre estos se definió la arquitectura cliente-servidor así como el MVC, Alta cohesión y Bajo acoplamiento. Se definió el diagrama de clases del diseño para el CU: "Mostrar estado del servidor", en la elaboración del mismo se dividió el sistema en paquetes funcionales correspondientes con la arquitectura en capas, obteniendo una capa para la presentación (Vista), una para la lógica de aplicación

Referencias

BARQUET, R.; J. RAMÓN, *et al.* Diseño de un prototipo de una cerradura inteligente para las viviendas bajo circunstancias determinadas, 2018.

CATALANI, E. *"Arquitectura Modelo/Vista/Controlador"*, 2007. [Disponible en: http://exequielc.wordpress.com/2007/08/20/arquitectura-modelovistacontrolador/

DUMÉNIGO, D. *Sistemas de información, aplicación en empresas. Contribuciones a la Economía*, 2012.

FRAMEWORK, E. P. *Concept: Use Case*, 2011. [Disponible en: http://epf.eclipse.org/wikis/openup/core.tech.common.extend_supp/guidances/concepts/use_case_BB199D1B.html?nodeId=e37dcf94

GARLAND, J. and R. ANTHONY. *Large-Scale Software Architecture*. Great Britain, John Wiley & Sons Ltd, 2003. p.

KO, J.; S. LEE, *et al. Real-time Mandatory Access Control on SELinux for Internet of Things*. 2019 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, USA, 2019. 1-6 p. 2158-4001

LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México, Prentice Hall, 1999. p.

MAR, O. and J. GULÍN Modelo para la evaluación de habilidades profesionales en un sistema de laboratorios a distancia *Revista científica*, 2018, 3(33): 332-343.

MAR, O.; I. SANTANA, *et al.* Competency assessment model for a virtual laboratory system and distance using fuzzy cognitive map *Revista Investigación Operacional*, 2017, 38(2): 170.178.

MÁRQUEZ, B. M. and J. M. ZULAICA. *Implementación de un reconocedor de voz gratuito a el sistema de ayuda a invidentes Dos-Vox en español*. Departamento de Ingeniería en Sistemas Computacionales Cholula, Puebla, México, Universidad de las Américas Puebla, 2004. p.

MARTÍN-PENA, D.; M. PAREJO-CUELLAR, *et al.* Las Tecnologías de la Información y la Comunicación en las radios universitarias españolas en el periodo 2012-2016 *Transinformação*, 2018, 30: 27-38.