PROYECTO FINAL DE CARRERA

Desarrollo e implementación de la herramienta de monitorización MERT

(Development and implementation of MERT monitoring tool)

Estudios: Ingeniería de Telecomunicaciones

Autor: Noel Castillo Mateo

Director: Ángel Viudez Martín

Director Ponente: Beatriz Otero Calviño

Año: 2016

Índice general

Colaboraciones VII
Agradecimientos IX
Resum del Projecte XI
Resumen del Proyecto XIII
Abstract XV
1. Introducción 1
1.1. Contexto del proyecto 2
1.2. Objetivos 4
1.3. Estructura de la memoria 5
2. Estudio previo 8
2.1. Aplicación equivalente 8
2.2. Herramientas y aplicaciones adicionales 9
2.2.1. Eclipse 9
2.2.2. Subversion 10
2.2.3. Apache Tomcat 11
2.2.4. Hudson 12
2.2.5. Weblogic 15
2.3. Comunicación con equipos remotos 17
2.3.1. SSH 17
2.3.2. FTP 17
2.3.3. JMX – t3 18
2.3.4. REST 19
2.4. Frameworks 20
2.4.1. Spring MVC 20
2.4.2. Hibernate 23
2.4.3. Maven 23
2.4.4. Junit 24
2.5. Lenguajes de programación 25
2.5.1. Java 25

2.5.2. Ant ... 26
2.5.3. Html5 ... 27
2.5.4. Css3 ... 28
2.5.5. JavaScript ... 28
2.6. Previsión inicial ... 29
2.7. Documentación funcional de la aplicación ... 31
2.7.1. Información versiones ... 31
2.7.2. Información entornos y estado instancias ... 33
2.7.3. Edición documento entornos ... 34
2.7.4. Aplicativo externo .jar ... 35
2.8. Documento técnico de la aplicación ... 35
2.8.1. Pantallas del aplicativo ... 36
3. Desarrollo de la aplicación ... 42
3.1. Arquitectura y estructura de la aplicación ... 42
3.2. Guía de navegación web ... 49
3.3. Implementación de la aplicación ... 50
3.3.1. Información versiones ... 50
3.3.2. Información entornos y estado instancias ... 55
3.3.3. Edición documento entornos ... 57
3.3.4. Aplicativo externo .jar ... 59
3.4. Job de Hudson y script de ANT ... 59
3.5. Requerimientos instalación ... 61
4. Pruebas de calidad ... 64
4.1. Test unitarios ... 64
4.2. Pruebas de rendimiento ... 66
5. Conclusiones ... 69
6. Anexo ... 72
6.1. SPS (Sistema de Promoción del Software) ... 72
6.2. Spring ... 73
6.3. Prototipos pantallas ... 75
6.4. Ciclo de Vida de un proyecto Maven ... 78
6.5. Pruebas de rendimiento ... 79
7. Referencias ... 85

Índice de figuras

Ilustración 2.1: Vista de versiones por cada entorno del aplicativo equivalente. 8
Ilustración 2.2: Nombres MQ de los Bindings 9
Ilustración 2.3: Entorno de desarrollo Eclipse. 10
Ilustración 2.4: Sistema de carpetas de *Subversion* Zeus 11
Ilustración 2.5: Esquema de conexión al Tomcat. 12
Ilustración 2.6: Pantalla de la herramienta de integración continua *Hudson* 13
Ilustración 2.7: Pantalla de configuración de una tarea del *Hudson* 14
Ilustración 2.8: Dominio de un Weblogic 15
Ilustración 2.9: Monitorización de *MBeans* en el nodo administrador mediante *Jconsole*. 16
Ilustración 2.10: Esquema de la arquitectura JMX. 18
Ilustración 2.11: Estudio de tecnologías Web más utilizadas. 21
Ilustración 2.12: Inclusión de dependencia en un archivo POM. 24
Ilustración 2.13: Ejemplo de código en Ant. 27
Ilustración 2.14: Ejemplo de página HTML. 27
Ilustración 2.15: Ejemplo de código CSSS introducido en una página HTML. 28
Ilustración 2.16: Primer diseño de pantalla de inicio. 29
Ilustración 2.17: Pagina de entornos por versión. 30
Ilustración 2.18: Histórico de versiones entorno. 30
Ilustración 2.19: Ejemplo de abstracción de un entorno. 31
Ilustración 2.20: Ejemplo de tabla de un entorno (*popup*) 32
Ilustración 2.21: Conexión JMX 33
Ilustración 2.22: Leyenda de estados de las instancias 34
Ilustración 2.23: Campos de la pantalla de información de entornos 34
Ilustración 2.24: Lista datos de un entorno 35
Ilustración 3.1: Estructura de la aplicacion en el SVN 42
Ilustración 3.2: Manera en que el archivo POM del padre define a sus módulos hijo que lo componen 43
Ilustración 3.3: Archivo *pom.xml* del proyecto padre 43
Ilustración 3.4: Petición a un modelo MVC. 44
Ilustración 3.5: Modelo MVC 44
Ilustración 3.6: Los sub-módulos de aux_service (izquierda) y de la capa de persistencia (derecha) 45
Ilustración 3.7: El sub-proyecto de la capa de servicios 46
Ilustración 3.8: Sub-módulo de la capa Web 47

Ilustración 3.9: Archivo de configuración de *Hibernate*. 48
Ilustración 3.10: Contexto de *Spring* de la capa de persistencia 48
Ilustración 3.11: Comunicación del aplicativo con las diferentes maquinas 49
Ilustración 3.12: Diagrama de flujo del aplicativo 50
Ilustración 3.13: Página de inicio, muestra las versiones por cada entorno 51
Ilustración 3.14: *Popup* de la página de inicio con la información completa de un entorno 51
Ilustración 3.15: Pagina inicial a la cual se le ha aplicado un filtro 'TEST1' 52
Ilustración 3.16: Dependencia JSCH para conectarse vía SSH 53
Ilustración 3.17: Código implementado para una conexión SSH 53
Ilustración 3.18: Dependencia commons-net utilizado para la conexión FTP 54
Ilustración 3.19: Implementación de la conexión vía FTP 54
Ilustración 3.20: Implementación mediante la librería 'wlfullclient.jar' de una conexión JMX al Weblogic 54
Ilustración 3.21: Pagina de las instancias a *Weblogic* 55
Ilustración 3.22: Directorio donde se guarda el archivo '*enviroment.txt*' que genera MERT 56
Ilustración 3.23: Libreria ml-ant-http utilizado para establecer conexión vía REST al *Hudson* 57
Ilustración 3.24: Clase encargada de lanzar el *job* del *Hudson* 57
Ilustración 3.25: Página de edición con los datos del entorno 'DESA1 58
Ilustración 3.26: Configuración del *job* del Hudson 60
Ilustración 3.27: Script de Ant 60
Ilustración 3.28: Paquete WAR del aplicativo 61
Ilustración 3.29: Aplicativo *Probe* desplegado en el *Tomcat* 61
Ilustración 3.30: Script para setear variables de entorno en Linux 62
Ilustración 4.1: Test de la parte de persistencia 65
Ilustración 4.2: Ejecución test 65
Ilustración 6.1: Arquitectura de Spring 73
Ilustración 6.2: Arquitectura básica de *Spring MVC* 74
Ilustración 6.3: Ciclo de vida de un *request* 75
Ilustración 6.4: Información versiones por entorno 76
Ilustración 6.5: Información del frontal por cada entorno 76
Ilustración 6.6: Información parte *Batch* de cada entorno 76
Ilustración 6.7: Información parte online de cada entorno 77
Ilustración 6.8: Información de las BBDD 77
Ilustración 6.9: Modo de edición 78
Ilustración 6.10: Informe agregado 1 usuario 79
Ilustración 6.11: Informe resultados en árbol 1 usuario 79
Ilustración 6.12: Informe árbol de resultados 1 usuario 'Edición TXT' 79
Ilustración 6.13: Informe árbol de resultados 1 usuario 'Información TXT' 80
Ilustración 6.14: Informe árbol de resultados 1 usuario 'Información Versiones' 80
Ilustración 6.15: Informe resultados en árbol 3 usuarios 80

Ilustración 6.16: Informe agregado 3 usuario 81
Ilustración 6.17: Informe reporte resumen 3 usuarios 81
Ilustración 6.18: Informe resultados en árbol 6 usuarios 81
Ilustración 6.19: Informe agregado 6 usuarios 81
Ilustración 6.20: Informe reporte resumen 6 usuarios 82
Ilustración 6.21: Informe resultados en árbol 10 usuarios 82
Ilustración 6.22: Informe agregado 10 usuario 82
Ilustración 6.23: Informe reporte resumen 10 usuarios 83
Ilustración 6.24: Informe reporte resumen 20 usuarios 83

Colaboraciones

Este proyecto ha sido desarrollado en el Departamento de Arquitectura de Everis Spain. Es un proyecto para la empresa GasNatural Fenosa.

Agradecimientos

En este punto, me gustaría dar las gracias a todas las personas que han contribuido directa o indirectamente en este trabajo.

En primer lugar me gustaría mostrar mi gratitud a todos mis compañeros del departamento de Arquitectura, por lo bien que me han tratado en estos meses. En especial me gustaría dar las gracias a Manuel Cruz y a Raúl Vázquez por haber dedicado tiempo a la supervisión del proyecto. En segundo lugar, estoy agradecido a mi directora ponente Beatriz Otero por confiar en mí y dedicarme su atención y sus consejos para finalizar con éxito el proyecto.

Por último, no me quiero olvidar de agradecer a mis amigos y familiares, porque me han ayudado a pasar aquellos momentos más complicados. En especial a mis padres, Manuel y Elena y a mi hermano, Kilian por su fe incondicional.

Resum del Projecte

Actualment existeixen un gran nombre de empreses encarregades de la generació de software, que normalment utilitzen una eina de gestió de versions per administrar-les de una manera eficient y ordenada. Quant més gran es el codi del projecte mes rellevant es torna la utilització d'aquest tipus d'eines. Un bon exemple es *Subversion* desenvolupada per Apache, que te com a objectiu principal gestionar les versions de qualsevol suport lògic de un sistema informàtic y emmagatzemar-les en un repositori remot.

El conjunt de versions de Zeus[1] es testejat en grups de maquines. Cada versió es desplega en un entorn diferent, caracteritzat per un conjunt de 3 maquines, dos de elles amb sistema operatiu Linux y la restant amb sistema operatiu Windows. Hi ha un total de 34 entorns per a provar Zeus.

En aquest projecte final de carrera, es dissenya e implementa MERT[2], una aplicació amb interfície Web que serveix per a mostrar la versió de Zeus que està desplegada en cada entorn.

MERT agilitza aquestes consultes mostrant les versions de tots els entorns, també les seves dades de configuració. Una consulta de MERT suposa 10 segons de carrega y es consulten les versions de 34 entorns.

Extreure la mateixa informació de manera manual pot suposar unes 3 hores, tenint en compte que la consulta a un entorn suposa entre 5 y 7 minuts.

[1]Zeus es la aplicació per a la gestió de la distribució de gas y electricitat de GasNatural en la que està col·laborant Everis.

[2]MERT es la aplicació amb interfície Web que es desenvolupa en el projecte final de Carrera. Les sigles fan referencia a Map Environment Release Tracer.

Resumen del Proyecto

Actualmente existen gran número de empresas encargadas de la generación de software, que normalmente utilizan una herramienta de gestión de versiones para administrarlas de una manera eficiente y ordenada. Cuanto más grande es el código del proyecto más relevante se hace la utilización de este tipo de herramientas. Un buen ejemplo es *Subversion* desarrollada por Apache, que tiene como objetivo principal gestionar las versiones de cualquier soporte lógico de un sistema informático y almacenarlas en un repositorio remoto.

El conjunto de versiones de Zeus[1] es testeado en grupos de máquinas. Cada versión se despliega en un entorno diferente, caracterizado por un conjunto de 3 máquinas, dos de ellas con sistema operativo Linux y la restante con sistema operativo Windows. Hay un total de 34 entornos para probar Zeus.

En este proyecto final de carrera, se diseña e implementa MERT[2], una aplicación con interfaz Web que sirve para mostrar la versión de Zeus que está desplegada en cada entorno.

MERT agiliza estas consultas mostrando las versiones de todos los entornos, también sus datos de configuración. Una consulta de MERT supone 10 segundos de carga y se consultan las versiones de 34 entornos.

Extraer la misma información de manera manual puede llevar unas 3 horas, teniendo en cuenta que la consulta a un entorno supone entre 5 y 7 minutos.

[1]Zeus es la aplicación para gestionar la distribución de gas y electricidad de GasNatural en la que está colaborando Everis.

[2]MERT es la aplicación con interfaz Web que se desarrolla en el proyecto final de carrera. Las siglas hacen referencia a Map Environment Release Tracer.

Abstract

Nowadays exist lots of companies which mission is generate software, normally they use a tool of version manage to administrate them in an orderly and efficient manner. As bigger as the code is it's more important to use this kind of tools. A good example is *Subversion* developed by Apache, whose main objective is to manage versions of any software of a computer system and store them in a remote repository.

The set of versions of Zeus is tested in groups of machines. Each version is deployed in a different environment characterized by a set of 3 machines, including 2 with Linux operating system and the remaining with Windows operating system. The amount of environments is approximately 34.

In this final career project, is designed and implemented an application called MERT[2], an application with Web interface that is useful on showing the version of Zeus that is deployed in each environment.

MERT shows in a quickly way the versions of all the environments, also all the configuration data. A charge of MERT cost about 10 seconds and the information read is from 34 different environments.

Read the same information in a manual way could take about 3 hours, because read each information of one environment represents between 5 and 7 minutes.

[1]Zeus is the application for managing the distribution of gas and electricity of GasNatural in which Everis is working.

[2]MERT is the application with Web interface that is developed in the final career project. The abbreviation refers to Map Environment Release Tracer.

1. Introducción

Este proyecto nace a partir de una beca de Innovación y Tecnología que empieza el 16 de setiembre de 2015 y acaba el 15 de enero de 2016 en la empresa Everis. El proyecto se desarrolla para Gas Natural Fenosa dentro del marco del proyecto Zeus.

Gas Natural es una empresa de distribución y comercialización de Gas. Para llevar a cabo sus funciones, la empresa dispone de múltiples aplicaciones. Zeus es la aplicación que se encarga de gestionar la distribución

Zeus está dividido en tres subsistemas principales (llamados EPS, ATEC, CCOM/PTEC) y otro subsistema transversal (llamado BASE) que aglutina los tres primeros. Esta división se lleva a cabo siguiendo patrones de funcionalidad. Por ejemplo, ATEC es la parte de Zeus que se encarga de la Atención TÉCnica.

La aplicación de Zeus está dividida en 4 capas: *Flex*, *WAR*, *Batch* y *Bindings*. Las tres primeras hacen referencia a la división MVC (Modelo, Vista, Controlador). *Flex* es la parte visual (la vista de la aplicación de Zeus), WAR es la capa de negocio (controlador) y *Batch* hace referencia a la capa de persistencia. En última instancia tenemos los *Bindings* que son enlaces a las características de cada región, legislación, tasas, etc. que son necesarios como parámetros de entrada para el aplicativo.

En el proyecto Zeus hay diferentes entornos donde se despliegan las diferentes aplicaciones. De esta manera se puede probar la funcionalidad de una versión determinada. Es en este contexto donde nace la idea de crear MERT. Esta extrae la información de la versión de las diferentes maquinas donde está desplegada cada una de las capas de la aplicación.

La información que se extrae de los entornos es vital a la hora de saber que mejora se está probando. Cada grupo de mejoras y/o correcciones se le asigna una versión, de tal manera que si se quiere probar una funcionalidad es necesario saber la versión de Zeus y de la librería

ADA que se está probando. De esta forma, se puede comprobar que lo que se está probando coincide con lo que se ha implementado en dicha versión.

En el caso en que se confunda la versión testeada con otra, esto supondrá una importante pérdida de horas de trabajo que se traduce en dinero para la empresa de Gas Natural Fenosa. Estaríamos hablando de unas pérdidas de unos 5.000€ a unos 10.000€ por una prueba. Aunque dependiendo de los equipos de trabajo involucrados podría ser mayor.

Existe una aplicación que se asemeja al funcionamiento de MERT. Esta aplicación tiene un coste computacional elevado, es por esto que solo se ejecuta una vez al día. Esto conlleva a que la información sea veraz a primera hora del día pero a medida que avanzan las horas puede no ser cierta.

MERT pretende ser una herramienta que agilice estas consultas y que en cualquier momento del día se pueda ejecutar. De esta manera se reducirían costes innecesarios para los equipos de testeo y demás que necesiten saber esta información.

1.1. Contexto del proyecto

Everis implementa una aplicación que se encarga de gestionar la distribución de gas y luz de la empresa cliente, Gas Natural Fenosa. En este proyecto hay más de 500 trabajadores que colaboran. Para ejemplificar la magnitud de las implementaciones, tenemos que: el subsistema de CCOM/PTEC ocupa 1,2 GBytes, EPS 340 MBytes, ATEC 300 MBytes y BASE 200 MBytes. De esta manera, el tamaño de cada versión que corresponde a la suma de los 4 subsistemas anteriores es de unos 2 GBytes, lo que se traduce en millones de líneas de código.

Al inicio del desarrollo del proyecto para gestionar la distribución del Gas (Zeus) no se pensó que fuera a alcanzar las dimensiones que tiene hoy en día. Por esta razón, el Sistema de Promoción del Software (SPS) que se creó estaba destinado a solventar problemas de un proyecto pequeño. En la medida en que el proyecto fue tomando nuevas dimensiones de desarrollo, el uso de SPS introdujo muchos problemas para gestionar las versiones.

Desde el inicio del proyecto Zeus se utilizó *Subversión* para administrar las versiones del software, aunque no se hizo de manera metódica y esto conllevo a problemas en el versionado. Para solventar este problema se creó un sistema de versionado de 4 dígitos (1.0.1.4-RC), de manera que se empezó a estructurar de forma más ordenada el código. Cada dígito hacía referencia a un tipo de software (correctivos, evolutivos, etc…) y las letras a un tipo de versión (definitiva, de prueba, etc…).

Posteriormente hubo problemas al crear copias del *trunk* (ramas o *branches*) y copias de las copias que no se reintegran con su propia rama padre sino con el *trunk* directamente. Todo esto origina problemas de árbol (*tree conflicts*) y dificultaba la reintegración de la versión. En algunos casos los problemas originados tenían un coste más elevado de lo que había supuesto el desarrollar el propio código de la versión. En otros casos, se han realizado testeos con versiones erróneas que no correspondían con los cambios que se querían probar.

Debido a estas experiencias se decidió adaptar el SPS tomando especial énfasis en el control del *Subversion*, añadiendo nuevas herramientas como MERT. (Ver más detalles de SPS en el Anexo 6.1).

La aplicación que se desarrolla en este proyecto (MERT) es una herramienta que permite realizar el control de todas las versiones, mostrará los datos de las rutas a las diferentes máquinas que corresponden a un entorno, así como también mostrará las rutas internas dentro de las máquinas y diferentes datos relevantes para la conexión con los servidores. La herramienta implementada añadirá nuevas funcionalidades tales como: mostrar el estado de cada instancia de *Weblogic* (ver 2.2.5) y permitir ejecutar una tarea de *Hudson* (ver 2.2.4) para la creación de los archivos *properties* (archivos requeridos para conectarse a los diferentes entornos).

1.2. Objetivos

El principal objetivo de la aplicación es mostrar la información desplegada en los diferentes entornos que manejan versiones del aplicativo. De esta manera, el usuario puede consultar la información veraz de manera ágil e intuitiva. Esto permitirá corregir errores tales como: consultar diferentes versiones para un mismo entorno o consultar otra versión del código. La herramienta a desarrollar ofrecerá también la posibilidad de consultar las versiones que están desplegadas en cada máquina.

MERT pretende ser una herramienta que aporte al proyecto Zeus las diferentes funcionalidades:

- Agilidad al consultar la información de los entornos.
- Control sobre las versiones.
- Control sobre el nivelado de entornos (misma versión en un entorno).
- Control sobre los subsistemas desplegados en un entorno.
- Manejabilidad a la hora de tratar con ‘environment.txt’ (creación y edición).

Para alcanzar estos objetivos, es necesario cumplir con los siguientes objetivos específicos:

- Analizar la problemática de la empresa y detectar las necesidades a incluir en la aplicación Web.
- Implementar un diseño de la aplicación donde se especifiquen las páginas que tendrá, el contenido y la funcionalidad de cada una de ellas.
- Evaluar y elegir la tecnología que se va a utilizar para desarrollar la aplicación.
- Identificar los servicios que se van a consumir para realizar la conexión con las diferentes maquinas (SO Windows y Linux).
- Estructurar la aplicación, decidir los *frameworks* i *plugings* que se van a utilizar.
- Elegir un entorno de desarrollo donde implementar la aplicación.
- Instalar todos los complementos necesarios para el desarrollo de la aplicación, buscar la compatibilidad entre los componentes.

- Diseñar e implementar las clases que van a formar parte de la aplicación, la parte de persistencia, los servicios que interactuaran con las máquinas externas y con las páginas web que visualizará el usuario final.
- Ejecutar los test para validar el correcto funcionamiento de la aplicación.
- Ejecutar las pruebas de rendimiento para comprobar el correcto funcionamiento en un entorno real de la aplicación.
- Maquetar las páginas web siguiendo el diseño de la empresa.
- Desplegar la aplicación en un servidor de aplicaciones y verificar que funcionan correctamente.
- Redactar un manual de usuario y una guía funcional que permita a terceros utilizar de forma sencilla la aplicación en sus entornos.

1.3. Estructura de la memoria

El documento está dividido en 4 capítulos. Hasta aquí, ya se ha descrito el primer capítulo que contiene la descripción del entorno de desarrollo y los objetivos del proyecto.

El segundo capítulo, analiza una aplicación con una funcionalidad similar, también se evalúan y se eligen las herramientas y la tecnología empleada para realizar la implementación de la aplicación. Además, el capítulo muestra los prototipos de pantalla y la documentación funcional y técnica de las 3 pantallas principales que describe el aplicativo.

El tercer capítulo describe como se ha llevado a cabo el desarrollo de la aplicación, considerando aspectos relacionados con la arquitectura utilizada y detalles de la implementación del código. En este capítulo, se explica el script de *Ant* que lanza una tarea del *Hudson* para la generación del fichero *properties* requerido por las demás tareas que lanzan los equipos del proyecto Zeus. Por último, se muestran los requerimientos de instalación de la aplicación MERT.

El cuarto capítulo incluye algunos test y muestra los resultados de rendimiento obtenidos para dichos test. Finalmente, el trabajo muestra las condiciones y líneas futuras de trabajo del proyecto.

2. Estudio previo

En este primer capítulo podremos encontrar el análisis de una aplicación similar a MERT, así como, las herramientas, lenguajes de programación, *frameworks* y protocolos utilizados en la implementación de la aplicación. También podremos encontrar los prototipos previos a la implementación y la documentación funcional y técnica del aplicativo.

2.1. Aplicación equivalente

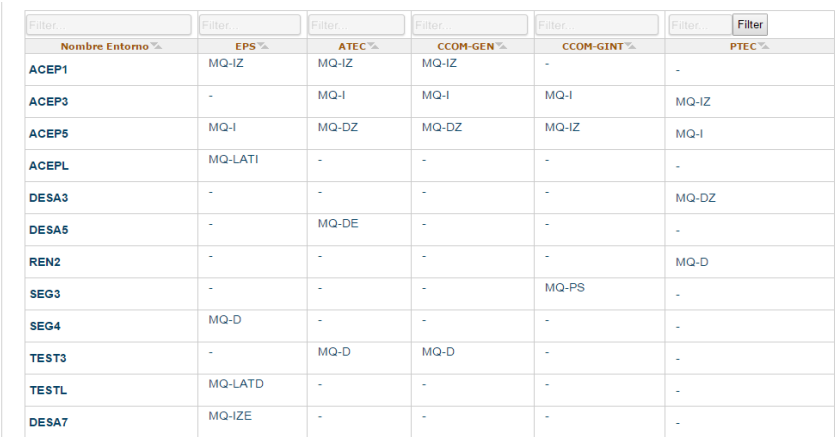
Existe una aplicación (IE) que muestra la misma información que MERT. La diferencia de esta aplicación respecto a la aplicación desarrollada en este proyecto radica principalmente en el camino que sigue para obtener la información.

La forma como se extrae la información en IE es un proceso lento. Esto es debido al alto coste computacional. Esta aplicación descomprime archivos de cada entorno donde se conecta y extrae la información de dentro del archivo comprimido. El tiempo que tarda en cargar la información es de entre unas 2 a 3 horas.

A continuación se muestra las dos pantallas de las que dispone la aplicación IE.

Nombre Entorno	Versión EPS	Versión ATEC	Versión CCOM	Versión cliente	Versión batch	Versión BBDD
ACEP1	4.3.1.0-RC-1 ada 5.9.1.0	4.3.1.0-RC-1 ada 5.9.1.0	4.3.1.0-RC-1 ada 5.9.1.0	4.3.1.0-RC-1	4.3.1.0 ada 5.9.1.0-RC-8	1.30
ACEP2	0.0.4.0-SNAPSHOT ada 5.9.1.0-RC-10	0.0.4.0-SNAPSHOT ada 5.9.1.0-RC-10	0.0.4.0-SNAPSHOT ada 5.9.1.0-RC-10	0.0.4.0-SNAPSHOT	0.0.4.0 ada 5.9.1.0-RC-8	1.41
ACEP3	4.3.0.0 ada 5.9.1.0	4.3.0.0 ada 5.9.1.0	4.3.0.0 ada 5.9.1.0	4.3.0.0	4.3.0.0 ada 5.9.1.0	1.156
ACEP4	4.3.1.0-RC-2 ada 5.9.1.0	4.3.1.0-RC-2 ada 5.9.1.0-RC-8	4.3.1.0-RC-2 ada 5.9.1.0-RC-8	4.3.1.0-RC-2	2.0.4.3 ada 5.9.1.0-RC-8	1.53
ACEP5	4.3.0.0 ada 5.9.1.0-RC-9	4.3.1.0-RC-4 ada 5.9.1.0-RC-4	4.3.0.0 ada 5.9.1.0-RC-9	4.3.1.0-RC-4	4.3.1.0 ada 5.9.1.0-RC-9	1.2
ACEP6	4.3.0.0 ada 5.9.1.0-RC-4	4.3.0.0 ada 5.9.1.0-RC-4	4.3.0.3 ada 5.9.1.0-RC-4	4.3.0.0	4.3.0.5 ada 5.9.1.0-RC-4	1.145
ACEP7	4.2.0.10 ada 5.9.0.1	- -	- -	4.3.0.6	4.2.0.10 ada 5.9.1.0-RC-4	-
ACEPL	4.3.0.6 ada 5.9.1.0-RC-4	- -	- -	4.3.0.0	4.3.0.6 ada 5.9.1.0-RC-4	1.100
DESA1	4.3.0.0-RC-8 ada 5.9.1.0-RC-4	- -	- -	0.0.3.0-SNAPSHOT	4.3.0.0 ada 5.9.1.0-RC-4	-
DESA3	0.1.5.3-SNAPSHOT ada 5.9.1.0-RC-8	0.1.5.3-SNAPSHOT ada 5.9.1.0-RC-8	0.1.5.3-SNAPSHOT ada 5.9.1.0-RC-4	0.1.5.3-SNAPSHOT	0.1.5.3 ada 5.9.1.0-RC-8ada-report-management-5.9.1.0-RC-8	-
DESA4	0.0.2.0-SNAPSHOT ada 5.9.1.0-RC-4	- ada 5.9.1.0-RC-4	0.0.2.0-SNAPSHOT ada 5.9.1.0-RC-4	2.0.2.0-SNAPSHOT	0.1.5.3 ada 5.9.1.0-RC-8ada-report-management-5.9.1.0-RC-8	-

Ilustración 2.1: Vista de versiones por cada entorno del aplicativo equivalente.



Nombre Entorno	EPS	ATEC	CCOM-GEN	CCOM-GINT	PTEC
ACEP1	MQ-IZ	MQ-IZ	MQ-IZ	-	-
ACEP3	-	MQ-I	MQ-I	MQ-I	MQ-IZ
ACEP5	MQ-I	MQ-DZ	MQ-DZ	MQ-IZ	MQ-I
ACEPL	MQ-LATI	-	-	-	-
DESA3	-	-	-	-	MQ-DZ
DESA5	-	MQ-DE	-	-	-
REN2	-	-	-	-	MQ-D
SEG3	-	-	-	MQ-PS	-
SEG4	MQ-D	-	-	-	-
TEST3	-	MQ-D	MQ-D	-	-
TESTL	MQ-LATD	-	-	-	-
DESA7	MQ-IZE	-	-	-	-

Ilustración 2.2: Nombres MQ de los *Bindings*

MERT corrige las deficiencias de esta aplicación que son: el tiempo de carga y la veracidad de los datos.

2.2. Herramientas y aplicaciones adicionales

Para la realización de este proyecto Web se requieren de herramientas y aplicaciones adicionales tanto para el propio desarrollo de la aplicación como para su funcionamiento. Como entorno de desarrollo interactivo se utiliza *Eclipse*. Integrado con *Eclipse* se utiliza *Subversion* que permite guardar remotamente el código. Por último, encontramos *Apache Tomcat* que sirve para el despliegue de la aplicación.

Como software externo utilizado para el correcto funcionamiento de MERT tenemos: *Weblogic* y *Hudson*.

2.2.1. Eclipse

Eclipse [1] es un IDE, entorno de desarrollo integrado, que está compuesto por un conjunto de herramientas de programación de código abierto y multiplataforma para el desarrollo de aplicaciones.

Este programa permite configurar el ambiente de desarrollo con base a las necesidades que se requieran en cada proyecto. Existen gran cantidad de *plugins* que nos brindan una mayor rango de funcionalidades para aplicaciones JEE, sin dejar de lado las aplicaciones de escritorio.

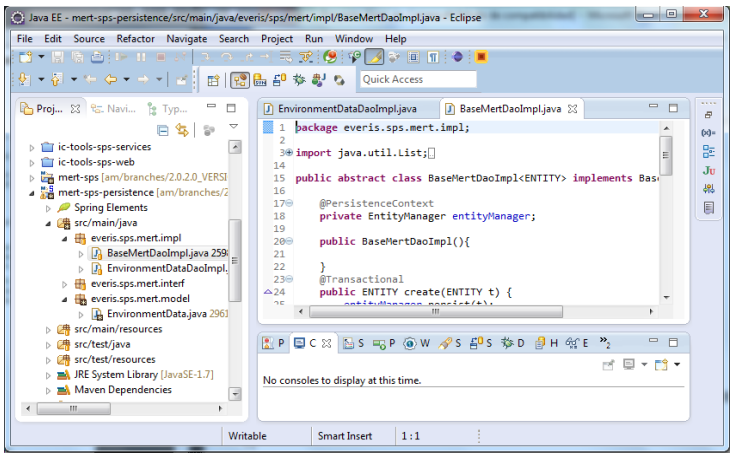


Ilustración 2.3: Entorno de desarrollo *Eclipse*.

Existen otros entornos de desarrollo muy utilizados pero en este proyecto se ha utilizado Eclipse por la cantidad de herramientas que se pueden descargar y la compatibilidad entre ellas.

La versión utilizada de *Eclipse* para el proyecto es la versión Juno, es decir, la 4.2.

2.2.2. Subversion

Subversion (SVN) [2] es el sistema de control de revisiones utilizado por la empresa para organizar y guardar el código del aplicativo Zeus. Es un software de código abierto basado en repositorios, el funcionamiento del cual se asemeja a un sistema de ficheros.

El *Eclipse* se conecta a un repositorio remoto mediante la dirección de éste. Entendiendo por repositorio un depósito o archivo centralizado donde se almacena y mantiene información digital, en nuestro caso archivos informáticos.

Para tener conexión con el repositorio remoto es necesario añadir su ruta HTTP. En la siguiente imagen (ilustración 2.4) tenemos la estructura de un repositorio de Zeus. Cualquier repositorio debe tener una estructura básica para facilitar su gestión. En el caso de Zeus se utilizan: *branches*, *tags* y *trunk*. El *trunk* es el código que se encuentra en producción, dentro de la carpeta *branches* cuelgan las versiones que se están desarrollando, por último, los *tags* son fotos de versiones para poder realizar pruebas.

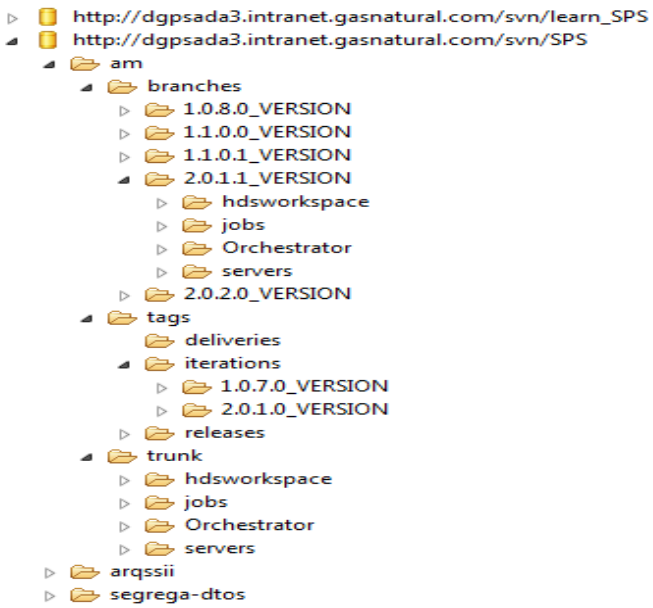


Ilustración 2.4: Sistema de carpetas de *Subversion* Zeus

2.2.3. Apache Tomcat

Apache Tomcat [3] funciona como un contenedor de *servlets* y es el escogido para desplegar la aplicación Web.

Tomcat es una herramienta de código abierto que implementa un *servlet Java*, *JavaServer Pages*, *Java Expression Language* y tecnología *WebSocket Java*.

En el caso del proyecto se utiliza la versión 7.0.57 porque es una versión que es compatible con los *frameworks* y herramientas utilizados. Se ha escogido esta tecnología debido a que dispone de los mínimos requisitos que permiten desplegar la aplicación. Al llevar mucho tiempo en el mercado posee una documentación extensa. También es fácil de utilizar.

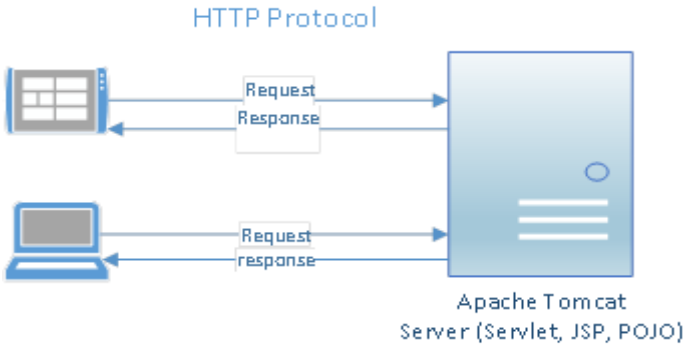


Ilustración 2.5: Esquema de conexión al Tomcat.

La principal función de *Apache Tomcat* es actuar como un servidor Web. Como vemos en la ilustración 2.5, un usuario lanza una petición HTTP al servidor *Tomcat* y este devuelve una respuesta. Hemos de tener en cuenta que el servidor *Tomcat* es el que tiene desplegada la aplicación y por tanto el que da respuesta a la petición Web.

2.2.4. Hudson

Hudson [4] es una herramienta de integración continua escrita en java que se ejecuta en contenedores de *servlets* como en *Apache Tomcat*. Trabaja con controladores de versiones como *Git*, *Subversion*, etc. y puede ejecutar proyectos basados en *Apache Ant* [5] y *Apache Maven* [6].

Es la herramienta que utiliza la empresa para tener control del código, ya sea para hacer actualización de código, borrar una rama, crear una rama desde el *trunk* y otras funcionalidades. Cada acción la hace un *job* en concreto, de manera que el programador si tiene permisos de ejecución puede hacer un *build* de dicha tarea.

En el proyecto Zeus normalmente estas tareas llaman a código *Ant*, el cual según los parámetros de entrada eliminará una rama u otra.

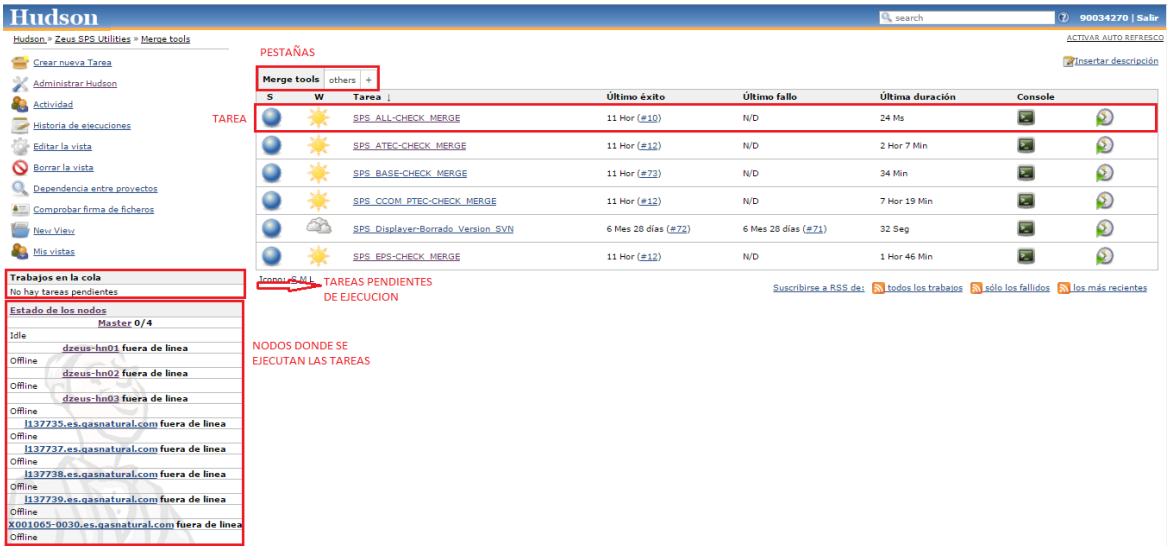


Ilustración 2.6: Pantalla de la herramienta de integración continua *Hudson*

Las tareas del *Hudson* se organizan en una vista como la que vemos en la anterior imagen (ilustración 2.6). Los grupos de tareas se organizan por pestañas. Por cada *job* se muestra el estado de la última ejecución, el estado de las últimas 5 ejecuciones, el tiempo que hace del último acierto, del último fallo y la duración de la última ejecución. Desde esta vista se puede abrir el *log* o ejecutar el *job*. En la parte inferior izquierda de la ilustración se muestran las tareas que están pendientes de ejecución, o en cola, y los nodos activos donde se ejecutan estas. El *Hudson* también permite diferentes funciones relacionadas con los nodos, la creación de tareas, edición de vista, etc.

En la ilustración 2.7 tenemos la ventana de configuración de una tarea. Los campos más determinantes aparecen marcados en rojo:

1. El nombre de la tarea.
2. Los usuarios que pueden interaccionar con el *job*, y que tipo de permisos tienen.
3. La versión de la JDK que utiliza el *job*.
4. El nodo en el que se va a ejecutar la tarea, en caso de estar vacío se ejecuta en el nodo master por defecto.
5. Los repositorios que se requieren descargar y en que ubicación.
6. Las ejecuciones que se van a producir. En el caso del ejemplo es una función programada con *Ant* con la versión 1.8.2.

Existe otro campo de interés, que es el de parámetros de entrada, aunque en este ejemplo no se utilizaba. Esto permite, una vez se ha seleccionado *build*, hacer una selección de parámetros de entrada previamente a la ejecución.

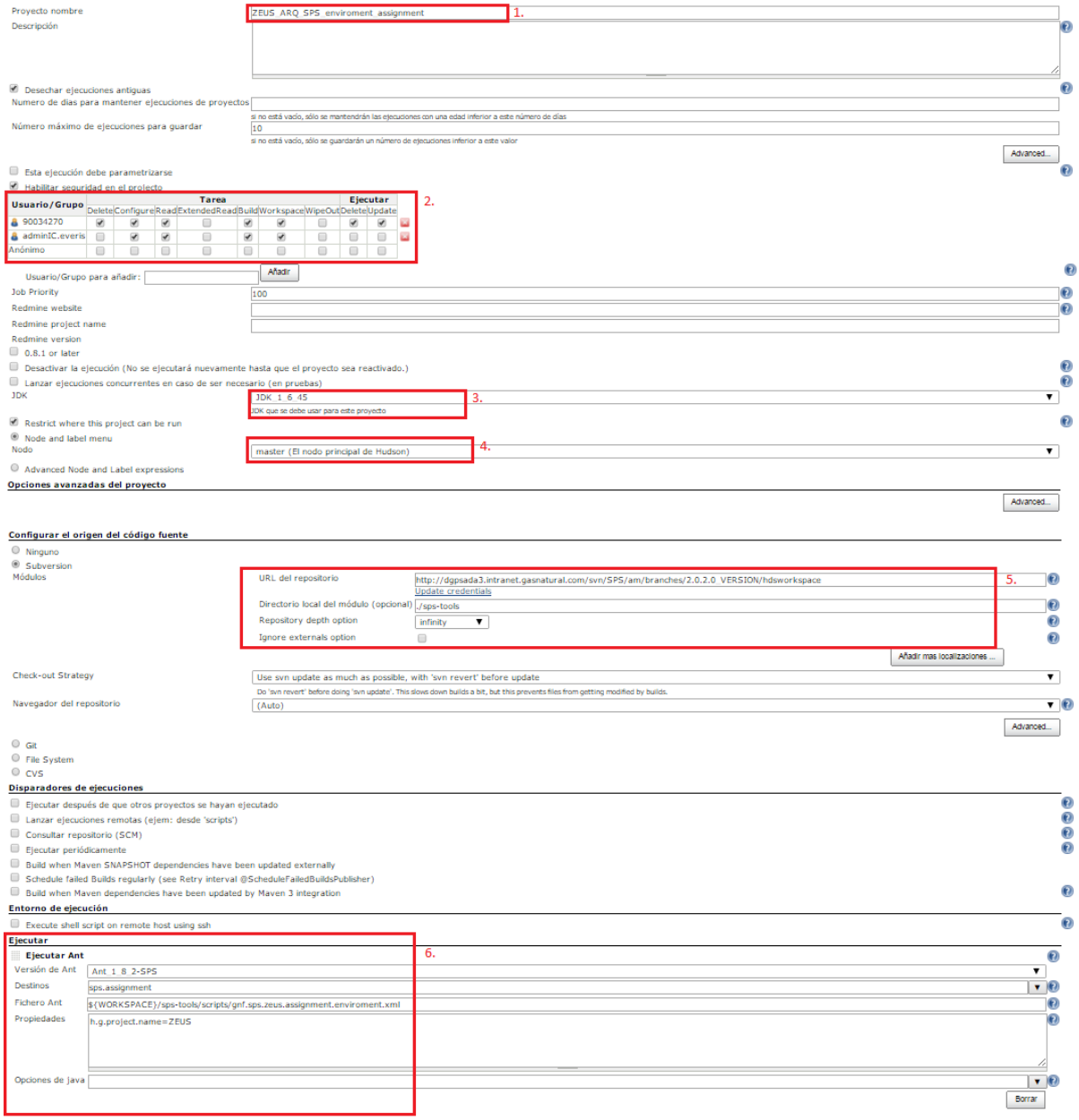


Ilustración 2.7: Pantalla de configuración de una tarea del *Hudson*

2.2.5. Weblogic

Weblogic [7] es un servidor de aplicaciones Java EE. Es una plataforma para aplicaciones multicapa distribuidas. En GasNatural es la que se utiliza para el despliegue de la capa *WAR* del aplicativo Zeus.

Cuando se hace un despliegue los ficheros de una aplicación Web se graban en una estructura de directorios que en nuestro caso se empaqueta en un solo fichero .war (*Web ARchive*). Un conjunto de descriptores de despliegue XML definen los componentes y los parámetros de ejecución de una aplicación, como también las configuraciones de seguridad.

Por así decirlo, el dominio del *Weblogic* es una colección de uno o más servidores (nodos) con sus respectivas aplicaciones y recursos de cada uno. Cada domino debe poseer una instancia de servidor especial diseñada para actuar como nodo administrador.

Para entornos de producción se utilizan múltiples instancias (*Managed Servers*) funcionando de manera independiente o en grupos llamados *clusters* (véase ilustración 2.8). El servidor de administración no hace de host de aplicaciones de producción.

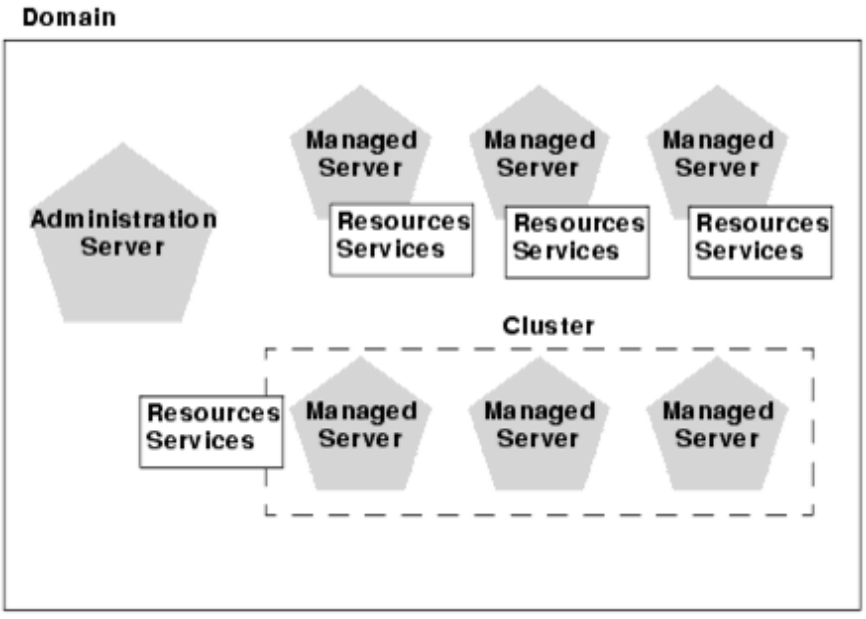


Ilustración 2.8: Dominio de un Weblogic

Otro punto importante son los *Beans*, son clases java con atributos privados y cada uno con sus respectivos métodos *getters* y *setters*, generalmente tienen un constructor vacío.

Dentro de los *Beans* existen otro tipo llamado *MBeans* (*Managed Beans*). Que son los encargados de gestionar el *Weblogic*, ya sean utilizados para la monitorización o para la configuración del mismo. Existen 3 tipos de *MBeans*:

- *Runtime MBeans*: Contienen información sobre el estado de las instancias y de sus recursos.
- *Configuration MBeans*: Contienen información sobre la configuración de los servidores y de los recursos.
- *Configuration MBeans for system modules*: Contienen información sobre la configuración de servicios como JDBC y JMS.

En el proyecto se ha utilizado el primer tipo de *MBeans*, porque permite obtener información sobre las instancias y sobre las aplicaciones desplegadas en ellas. El aplicativo MERT utilizara JMX (*Java Management Extensions*) es una tecnología Java que me permitirá conectarse a estos *Mbean* para extraer la información.

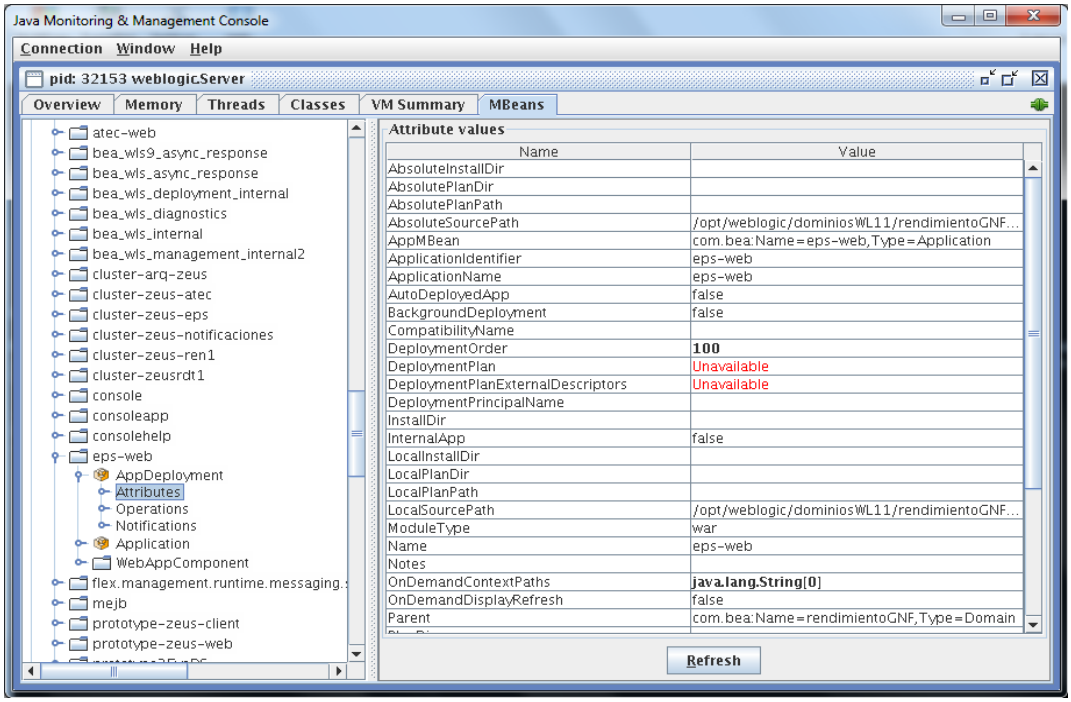


Ilustración 2.9: Monitorización de *MBeans* en el nodo administrador mediante *Jconsole*.

En la ilustración 2.9, mediante una herramienta de monitorización incluida en el JDK[8], se ve la estructura de una instancia. Para leer esta información ha sido necesario conectarse mediante el protocolo SSH al nodo administrador y ejecutar el comando de la herramienta *Jconsole*, donde se ha seleccionado el proceso que corresponde con el nodo administrador.

2.3. Comunicación con equipos remotos

En este apartado podremos encontrar las diferentes comunicaciones que se han utilizado para recoger los diferentes datos de los entornos.

2.3.1. SSH

SSH (*Secure SHell*)[9] es el nombre que recibe un protocolo que se utiliza para acceder a maquinas remotas a través de una red. Permite manejar de forma completa la maquina remota mediante un intérprete de comandos, en caso de ser requerido también se puede redirigir el tráfico a un servidor X para visualizar de manera gráfica el terminal.

SSH establece un túnel hacia el servidor remoto. El cliente y el servidor utilizan firmas digitales para verificar su identidad. Lo cual dificulta a un terminal desconocido escuchar el tráfico. Este tipo de comunicación es el que se utilizarán en la aplicación MERT para la conexión a la parte *Batch* de los entornos y a las máquinas donde está instalado el *Weblogic*. Debido a que son máquinas con sistema operativo *Linux* y permiten este tipo de comunicación.

2.3.2. FTP

FTP[10] es un protocolo de transferencia de archivos, su objetivo es transmitir exitosamente archivos entre máquinas en una red sin que el usuario tenga que iniciar sesión en el host remoto. Es por lo tanto un protocolo inseguro.

Existen dos tipos de modos de funcionamiento el activo y el pasivo:

- Modo activo: Es el método original en el que el cliente FTP inicia la transferencia de datos, el servidor abre una conexión desde el puerto 20 para la dirección IP y un puerto aleatorio sin privilegios (mayor que 1024) especificado por el cliente. Debido a la existencia de internet muchos cortafuegos rechazan las peticiones a puertos mayores que el 1024. Por este motivo utilizare el modo pasivo, debido a la imposibilidad de transferencia de ficheros.
- Modo pasivo: El proceso es similar, el cliente FTP indica que quiere acceder a los datos de modo pasivo y el servidor proporciona la dirección IP y el puerto aleatorio mayor que 1024 sin privilegios. Posteriormente, el cliente se conecta al puerto en el servidor y descarga la información requerida. Y de esta manera el cliente resuelve la interferencia del cortafuegos en el lado cliente.

2.3.3. JMX – t3

Java Management Extensions (JMX)[11] es una tecnología que permite administrar y/o supervisar aplicaciones, objetos del sistema, dispositivos y redes orientadas a servicio. La entidad administradora/supervisada se representa mediante objetos llamados *MBean*.

JMX es la tecnología utilizada para comunicarse con los *Weblogic* que se administran mediante *MBeans*.

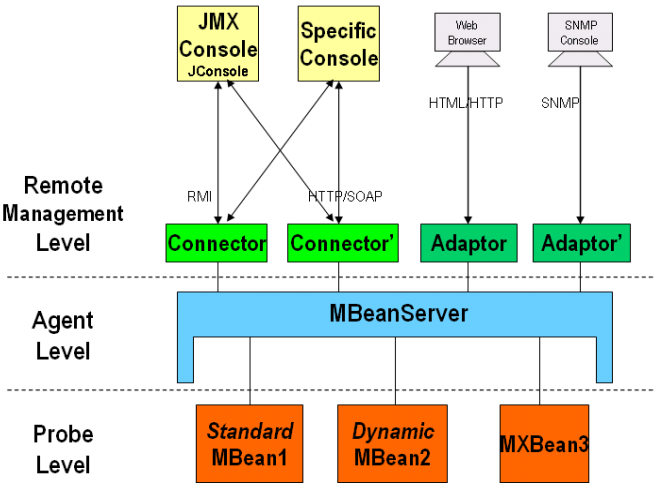


Ilustración 2.10: Esquema de la arquitectura JMX.

La arquitectura JMX está estructurada en 3 capas como se ve en el esquema 2.10.

- La capa de instrumentación: Determina como se pueden envolver los recursos dentro de los *MBeans*.
- La capa de agente: Consta del servidor de *MBeans* y de agentes, que proporcionan una infraestructura de gestión. Los servicios que se implementan son:
  - Supervisión
  - Notificación de sucesos
  - Temporizadores
- La capa de gestión: Define como pueden interactuar las aplicaciones de gestión externas.

Los conectores que se utilizan para conectar un agente JMX a una aplicación de gestión remota habilitada para JMX pueden ser los siguientes:

- Conector SOAP JMX
- Conector RMI-IIOP JMX o RMI-T3 JMX

En el aplicativo MERT se utiliza el conector RMI-T3 JMX[12]. Este conector se utiliza para transmitir datos entre el servidor de *Weblogic* y otros programas *Java*, incluyendo clientes y otras instancias de *Weblogic*. El *Weblogic* mantiene la traza de cada JVM (*Java Virtual Machine*) conectada a la aplicación. Para llevar información a la JVM el *Weblogic* crea una sola conexión T3. Este tipo de conexiones maximizan la eficiencia eliminando múltiples protocolos usados para comunicar entre redes, utilizando menos recursos del sistema y también minimizando los tamaños de paquete (que aumenta la velocidad de entrega).

2.3.4. REST

Existen dos grandes tipos de servicios Web: los servicios Web SOAP[13] y los servicios Web REST[14].

- SOAP (*Simple Object Access Protocol*): Es un protocolo estándar que define como dos objetos de diferentes procesos pueden comunicarse por medio de intercambios de datos xml. Es una especificación de protocolo para intercambiar información estructurada. Se le pueden aplicar extensiones de seguridad, routing, puede ser usado sobre diferentes protocolos de transporte y permite cualquier modelo de programación. La contrapartida es que es más costoso utilizarlo, implementa más funcionalidades.

- REST (*REpresentational State Transfer*): Es un estilo de arquitectura de software que utiliza *World Wide Web*. Se comunica por medio del protocolo HTTP.

REST es más simple que SOAP, debido a que SOAP añade más funcionalidades. Por este motivo se ha escogido REST en la aplicación MERT ya que solo se ha de hacer una petición HTTP de la cual no se ha de recoger el resultado. La tarea en la que se necesita es en la ejecución del *job* de *Hudson*.

2.4. Frameworks

En este apartado se tratan los *frameworks* utilizados en la aplicación MERT y los motivos de su elección.

2.4.1. Spring MVC

Spring MVC[15] es uno de los módulos del *framework* de Spring y como su propio nombre indica implementa una arquitectura Modelo – Vista – Controlador.

Comparativa otros frameworks

Los *frameworks* Web más usados como vemos en las siguiente grafica son *Spring MVC*, *JSF*[16] y *Vaadin*[17].

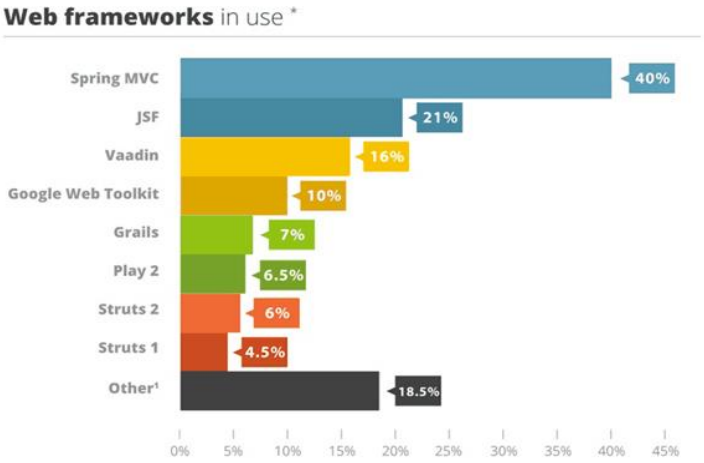


Ilustración 2.11: Estudio de tecnologías Web más utilizadas.

- **Spring MVC**

Spring MVC contiene muchas herramientas, como por ejemplo *Spring Tool Suite*, todas las dependencias *Maven* disponibles en su repositorio público. Eclipse incluye también soluciones extra como con *MyEclipse* que añade herramientas para *Spring*.

Las aplicaciones Spring son altamente escalables. Ya que el *framework* está pensado para aplicaciones de alta usabilidad. Disponen de una amplia documentación que cubre todo. Por otra parte la página oficial ofrece múltiples tutoriales.

En Spring MVC no existen prototipos para construir aplicaciones de manera rápida y fácil. Es una tecnología difícil de manejar al inicio por este motivo. Por otro lado es una herramienta abierta al ensamblaje con otros *frameworks*.

No proporciona ningún *framework* para construir interfaces de usuario.

- **Vaadin**

Es fácil de desarrollar, permite arrastrar componentes en una pantalla y proporciona la lógica de cada componente. Existen muchos tutoriales y guías para aprender rápido a trabajar

con Vaadin. Además, permite descargar componentes de un repositorio, los cuales pueden ser usados para desarrollar (*the Vaadin Directory*).

Vaadin se caracteriza por tener pobre escalabilidad desde que guarda el estado y la lógica en el servidor, eso quiere decir, que para cada usuario se necesita enviar al servidor la petición, y el cliente tiene que esperar para saber cómo reaccionar. Esto produce un tráfico y un tiempo de carga mayor.

- **JSF**

Forma parte del Standard de Java EE. Por otra parte es fácil reutilizar los componentes cosa que provoca que sea fácil y rápido crear una implementación.

El problema que tiene es que es un *framework* cerrado, y por este motivo no soporta muy bien trabajar con otros *frameworks*.

Finalmente el *framework* utilizado en la aplicación ha sido *Spring MVC*. Esto es debido a que permite una fácil adaptación con otros *frameworks*. Por lo tanto, es más flexible. No genera componentes ya creados, por lo que permite un control más detallado del código que se está ejecutando, y alta escalabilidad. Por estos motivos se ha seleccionado *Spring MVC*, también teniendo en cuenta la amplia documentación de la que disponemos como los innumerables ejemplos y tutoriales de ayuda disponibles.

Para más detalles sobre Spring y sus módulos véase el apartado 6.2 del anexo. Cabe destacar que Spring MVC trabaja con JSP que es un lenguaje HTML dinámico.

2.4.2. Hibernate

Hibernate[18] es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java. Este *framework* facilita el mapeo de atributos entre la base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos XML y anotaciones.

Hibernate es un *framework* para persistir objetos en una base de datos relacional, es decir, una solución para no tener que tratar directamente con la base de datos. Además proporciona:

- Productividad: Evita mucho código confuso en la capa de persistencia, permitiendo centrarse en la lógica de negocio.
- Mantenibilidad: Gracias a tener pocas líneas de código permite que sea más claro y fácilmente modificable.
- Rendimiento: Ofrece el mismo rendimiento que puede ofrecer una solución "manual".
- Independencia del proveedor: Es una solución ORM y esto permite abstraerse del SGBD (Sistema de Gestión de Bases de Datos) y desarrollar en local, con base de datos ligeras sin implicación en el entorno de producción.

Hibernate es el framework utilizado en MERT para tratar con la base de datos de una manera más ágil. Se trabaja con MySQL. MySQL es un sistema de gestión de base de datos relacional, *multithread* y multiusuario. Dispone de un lenguaje propio que no necesitamos conocer al trabajar con *Hibernate*.

2.4.3. Maven

Maven (MVN) [19] es una herramienta destinada a la gestión y construcción de proyectos. Es el *framework* que se utiliza en la construcción de MERT por diferentes motivos, pero principalmente por las llamadas dependencias. *Maven* permite la descarga de librerías mediante la modificación del archivo de configuración (POM: *Project Object Model*). Este archivo llamado pom.xml, es donde se estructura todo el proyecto. Al incluir estas librerías en el archivo pom.xml *Maven* se descarga de manera automática de un repositorio remoto la

versión indicada, y no solo eso si un paquete descargado depende de otro se descarga también dicha librería. Todo esto simplifica la descarga de librerías.

A continuación en la ilustración 2.12 mostramos un buen ejemplo que ilustra la descarga de una dependencia, donde se indica el grupo, el nombre del artefacto y la versión del mismo.

```
<dependency>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
   <version>1.2.17</version>
</dependency>
```

Ilustración 2.12: Inclusión de dependencia en un archivo POM.

Maven ayuda a crear proyectos desde cero, ya que dispone de unos "arquetipos" que son plantillas que sirven para crear un esqueleto completo de manera automática, en función de los parámetros que se le indique.

En último lugar, *Maven* es una herramienta que permite configurar el entorno, crear una estructura padre-hijo y ejecutar test unitarios. Se puede leer más sobre *Apache Maven* en el anexo 6.4.

2.4.4. Junit

Junit [20] es un *framework Java* que permite la ejecución de clases de manera controlada, para poder comprobar que los métodos realizan su cometido de forma correcta.

También sirve como herramienta para realizar las pruebas de regresión, que se realizan cuando una parte del código ha sido modificada y sea necesario comprobar que se sigue cumpliendo con todos los requisitos.

2.5. Lenguajes de programación

A continuación describimos brevemente los lenguajes de programación utilizados en MERT.

2.5.1. Java

Java[21] es un lenguaje de programación orientado a objetos. El principal propósito de la creación de Java fue crear un lenguaje con el que el usuario pudiese interactuar en internet y que se pudiera ejecutar en cualquier sistema operativo. Es el principal lenguaje que se va a utilizar en la aplicación. Sus principales características son:

- Lenguaje totalmente orientado a objetos: Paradigma de programación que usa objetos en sus interacciones. Incluye técnicas de herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.
- Disponibilidad de un amplio conjunto de bibliotecas.
- Lenguaje simple: Desde el principio es fácil crear una aplicación. Utiliza conceptos de C++ y C.
- Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando asi la creación de aplicaciones distribuidas.
- Interpretado y compilado a la vez: Java es compilado, en la medida en que se transforma en una especie de código máquina, los *bytecodes*, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los *bytecodes* se pueden ejecutar directamente sobre cualquier maquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real.
- Robusto: Java fue diseñado para ser fiable. Ha prescindido de los punteros los cuales pueden dar lugar a problemas de memoria y posee un recolector de basura.
- Seguro: Se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

- Indiferente a la arquitectura: El compilador de Java genera *bytecodes*: un formato intermedio indiferente a la arquitectura diseñado para ser transportado a multiplex plataformas. El resto lo soluciona el intérprete de Java.
- Portable: Java especifica los tamaños de sus datos básicos y el comportamiento de sus operadores aritméticos, los programas son iguales en todas las plataformas. Debido a la Máquina Virtual de Java (JVM).
- Alto rendimiento.
- *Multithread*: Permite ejecutar más de un hilo a la vez.
- Dinámico: El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.
- Produce *applets*: Pequeños programas que aparecen integrados en las páginas Web.

2.5.2. Ant

Apache Ant[5] es una herramienta usada en programación para realizar tareas mecánicas y repetitivas normalmente relacionadas con la fase de compilación y construcción. Los archivos han de estar escritos en XML. Cada build.xml contiene un proyecto (*project*) y al menos un objetivo (target). Cada target puede contener varias tareas (*task*) que son fragmentos de código a ejecutar. Cada propiedad consta de un nombre y un valor y son usados para asignar valores a las tareas.

```
<project name="MyProject" default="dist" basedir=".">
   <description>
       simple example build file
   </description>
 <property name="src" location="src"/>
 <property name="build" location="build"/>
 <property name="dist"  location="dist"/>
 <target name="init">
   <tstamp/>
   <mkdir dir="${build}"/>
 </target>
 <target name="compile" depends="init" description="compile the source " >
   <javac srcdir="${src}" destdir="${build}"/>
 </target>
 <target name="dist" depends="compile" description="generate the distribution" >
   <mkdir dir="${dist}/lib"/>
   <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
 </target>
 <target name="clean" description="clean up" >
   <delete dir="${build}"/>
   <delete dir="${dist}"/>
 </target>
</project>
```

Ilustración 2.13: Ejemplo de código en Ant.

2.5.3. HTML5

HTML5[22] es la última versión de lenguaje de marcado de hipertexto para los sitios Web desarrollados por *World Wide Web Consortium* (W3C).

HTML5 añade etiquetas para manejar la semántica: *header*, *footer*, *article*, etc. Estas etiquetas permiten describir cual es el significado del contenido (ver ilustración 2.14).

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>fuente de múltiples elementos</title>
  </head>
  <body>
    <audio id="audioTestElem" autobuffer controls >
      <source src="test.m4a">
      <source src="test.ogg" type="audio/ogg; codecs=vorbis">
      <source src="url">
      no audio for you
    </audio>
  </body>
</html>
```

Ilustración 2.14: Ejemplo de página HTML.

2.5.4. Css3

CSS (*Cascading Style Sheets*) [23] es un lenguaje que permite añadir elementos de presentación a un documento estructurado escrito en HTML o XML. Es decir, se utiliza para dar estilo a documentos HTML o XML, separando el contenido de la presentación (ver un ejemplo en la ilustración 2.15).

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: #d0e4fe;
}
h1 {
    color: orange;
    text-align: center;
}
p {
    font-family: "Times New Roman";
    font-size: 20px;
}
</style>
</head>
<body>
<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Ilustración 2.15: Ejemplo de código CSSS introducido en una página HTML.

2.5.5. JavaScript

JavaScript (JS)[24] es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Sirve para crear pequeños programas que luego son insertados en una página Web y en programas más grandes, orientados a objetos mucho más complejos. Con *JavaScript* se puede crear diferentes efectos e interactuar con los usuarios, aporta dinamismo a la página Web HTML.

2.6. Previsión inicial

La aplicación pretende ser una herramienta de monitorización donde por cada equipo de trabajo se visualice información de los entornos que tiene asignados y las versiones que tiene desplegadas en cada entorno. También muestra un histórico de los entornos por los que ha estado cada versión.

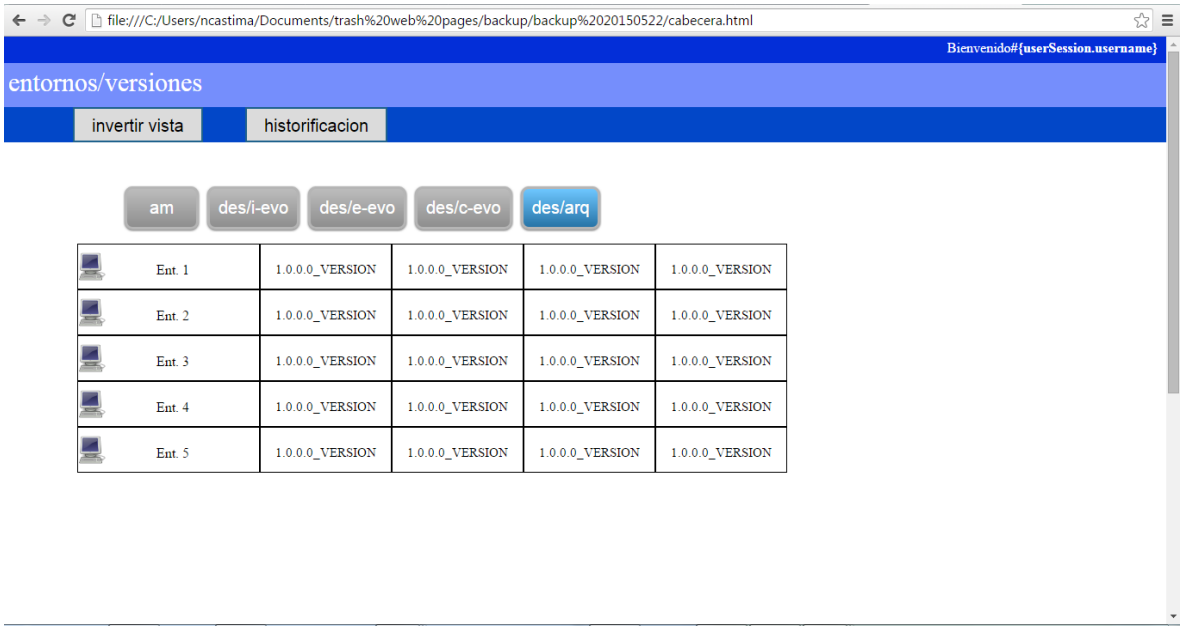


Ilustración 2.16: Primer diseño de pantalla de inicio.

La ilustración 2.16 muestra la primera pantalla donde se puede acceder a los entornos disponibles para cada equipo de trabajo, en cada uno aparece la lista de versiones desplegadas. La versión activa aparece resaltada respecto el resto de versiones.

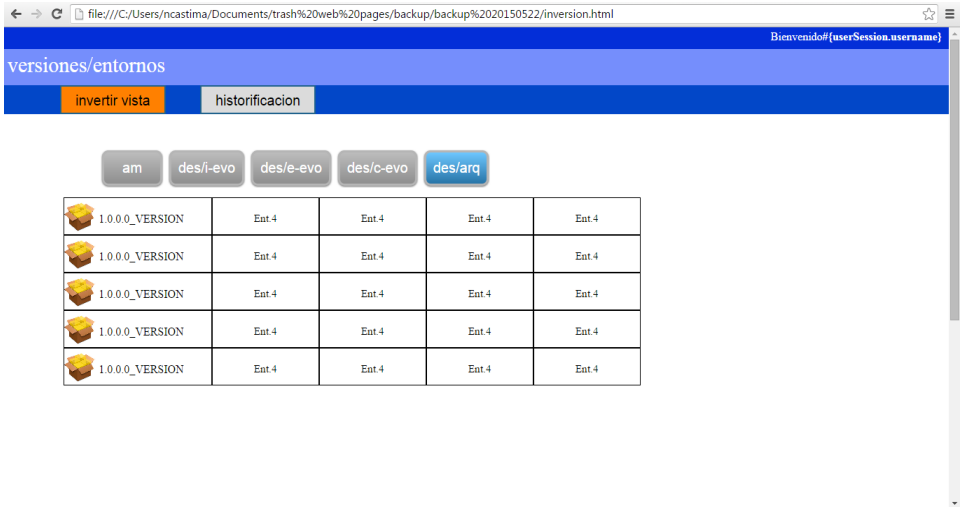


Ilustración 2.17: Pagina de entornos por versión.

En la segunda pantalla (ver ilustración 2.17), el aplicativo permite filtrar según el equipo de trabajo. Dependiendo del equipo de desarrollo se muestran las versiones que dependen de él y los lugares donde está desplegada cada una.

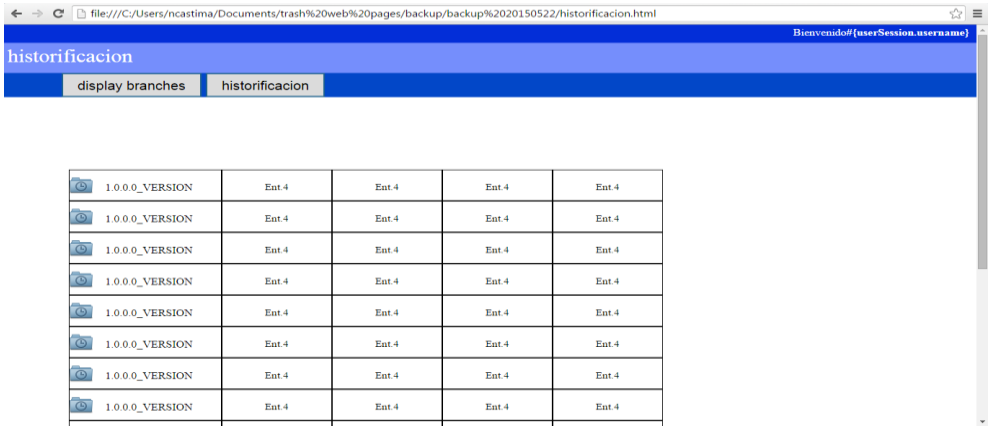


Ilustración 2.18: Histórico de versiones entorno.

Por último, en la ventana de histórico se muestra por qué entornos ha pasado cada versión (ver ilustración 2.18).

Posteriormente a este frontal se decidió en el equipo de arquitectura que datos se deberían modificar para hacer más útil el aplicativo y se hizo un diseño más acorde al definitivo. En el anexo 6.3 se muestran otros prototipos sobre la aplicación.

2.7. Documentación funcional de la aplicación

En este apartado podremos encontrar las funciones que va a aportar cada pantalla, la información disponible y como el usuario puede interaccionar con ella.

2.7.1. Información versiones

La página de "Información Versiones" es la ventana donde se muestra la información de las versiones de Zeus. La información que aparece por cada entorno es la siguiente: el alias del entorno, el equipo desarrollo asignado, los subsistemas que participan en el despliegue, las versiones de Zeus, BASE y de ADA, y los *Bindings*.

Para saber que subsistema está desplegado se marca con dos colores el azul (se ha realizado el despliegue del subsistema) y el gris (el subsistema no está desplegado).

Las versiones de Zeus, BASE y ADA se comparan las que se han utilizado en los diferentes subsistemas y en las diferentes máquinas de un entorno. En el caso de coincidir se escribe el nombre de la versión, si no coincide aparece el mensaje: 'BAD_VERSION'.

En último lugar se muestran los *Bindings* activos (verde), inactivos (gris) y erróneos (rojo). Los *Bindings* pueden llegar a ser erróneos si su constante MQ no coincide entre la parte desplegada en la maquina *Batch* y la de *Weblogic* (Véase la ilustración 2.19).

ENTORNO	EQUIPO	ATC	EPS	PTC	ZEUS	BASE	ADA	B ATE	B EPS	B PTC
REN1	EVERIS				4.2.2.2-SN-0	4.2.2.2-SN-0	4.2.2.2-SN-0			
REN1	EVERIS				4.2.2.2-SN-0	4.2.2.2-SN-0	4.2.2.2-SN-0			

Ilustración 2.19: Ejemplo de abstracción de un entorno.

La aplicación MERT permite presionar encima de un entorno, esto provoca que aparezca un *popup* (ventana emergente) donde se muestra la información de las versiones de los diferentes artefactos desplegados en un entorno.

ALIAS ENTORNO	EQUIPO	EPS		PTCCOM		ATEC		Bindings			
REN1	EVERIS	EPS	4.2.2.2-SN-0	ptccom	4.2.2.2-SN-0	atec	4.2.2.2-SN-0	EPS	MQ-I	CCOM-GEN	MQ-IZ
BATCH		BASE	4.2.2.2-BE-1	BASE	4.2.2.2-BE-1	BASE	4.2.2.2-BE-1	ATEC	MQ-IZ	CCOM-GINT	-
		ADA	4.2.2.2-RC-2	ADA	4.2.2.2-RC-2	ADA	4.2.2.2-RC-2			PTEC	-
		EPS	4.2.2.2-SN-0	ptccom	4.2.2.2-SN-0	atec	4.2.2.2-SN-0	EPS	MQ-I	CCOM-GEN	MQ-IZ
ONLINE		BASE	4.2.2.2-BE-1	BASE	4.2.2.2-BE-1	BASE	4.2.2.2-BE-1	ATEC	MQ-IZ	CCOM-GINT	-
		ADA	4.2.2.2-RC-2	ADA	4.2.2.2-RC-2	ADA	4.2.2.2-RC-2			PTEC	-
FRONTAL		zeus	4.2.2.2-BE-1	ADA	4.2.2.2-BE-1	BASE	4.2.2.2-BE-1				

Ilustración 2.20: Ejemplo de tabla de un entorno (*popup*)

En la tabla (ilustración 2.20) se muestra el alias, el equipo de desarrollo, las versiones de los subsistemas EPS, PTCCOM y ATEC (la del propio subsistema, BASE y ADA) y en último lugar los 5 *Bindings* (EPS, ATEC, CCOM-GEN, CCOM-GINT, PTEC).

Cada subsistema tiene una parte desplegada en una maquina *Linux* donde se encuentra la capa *Batch* de la aplicación, otra parte en una maquina *Windows* donde está desplegada la capa frontal o *Flex* de la aplicación y por último, su capa WAR está desplegada en un *Weblogic*. Cada una de las capas debería estar usando la misma versión sino la aplicación no tendría un funcionamiento correcto.

Al presionar con el botón izquierdo sobre el alias de un entorno se accede al balanceador que selecciona el frontal web. De esta manera, se abre una pestaña nueva con la URL de Zeus.

Para extraer la información de las versiones MERT se conecta por un lado al *Weblogic* para consultar las versiones de los 3 subsistemas, además lee cada uno de las versiones de BASE y las versiones de ADA. Como se ve en la siguiente imagen se conecta utilizando el protocolo t3 mediante un cliente JMX.

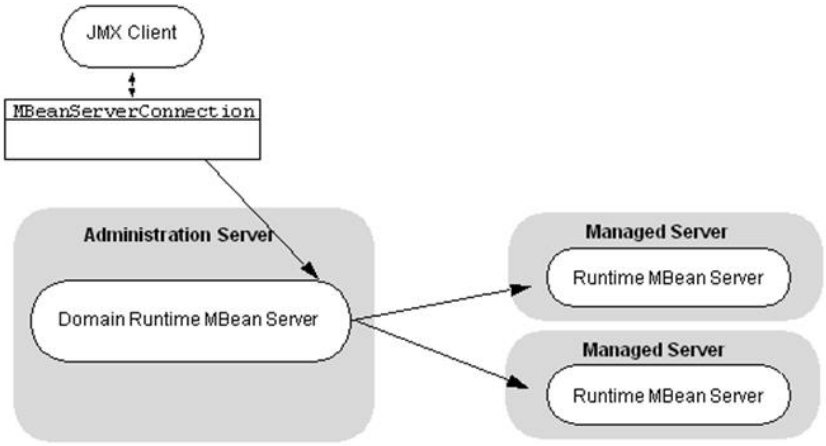


Ilustración 2.21: Conexión JMX

La información de las versiones de los *Batch* se extrae mediante una conexión remota utilizando el protocolo SSH, leyendo de un fichero interno. En el caso de la versión desplegada en el frontal web (ISS) se utiliza una conexión mediante el protocolo FTP, en este caso también se lee un fichero en la maquina destino. Para leer la información de los *Bindings* se utiliza el protocolo SSH donde se lee un archivo *properties* que contiene la información. Cada una de las consultas a las versiones se ejecutan cada vez que se actualiza o se carga la página.

Esta página implementa un filtro, gracias al cual según unos datos de entrada comprueba las coincidencias en los parámetros de la tabla, si algún entorno no posee tales coincidencias se oculta.

2.7.2. Información entornos y estado instancias

En la pantalla de 'información de *Batch*, *Weblogic* y *Flex'* se muestran el alias del entorno, el equipo de desarrollo, la ruta de la maquina donde se despliega el *Batch*, la ruta donde está desplegado, la URL de la parte Flex y el directorio de la ruta donde está desplegado, y el servidor de *Weblogic* y el *server name* de cada instancia, más un campo que marca el estado del servicio. Puede haber hasta 3 instancias por cada entorno en el *Weblogic* con su respectivo recuadro que indica el estado. Hay 6 posibles estados de las instancias de *Weblogic* que se detallan a continuación en la ilustración 2.22:

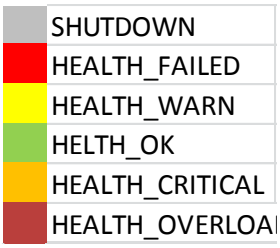


Ilustración 2.22: Leyenda de estados de las instancias

La información se extrae de la base de datos, exceptuando el estado de los servidores del *Weblogic* a los cuales mediante un cliente JMX se lee su estado. También se dispone de un botón para generar un archivo (‘environment.txt’) con los datos de todos los entornos, junto con otro que lanza una tarea del *Hudson* que ejecuta un script que analiza el archivo y genere ‘n’ archivos *properties*, con ‘n’ igual al número de entornos. Este script analiza el archivo y extrae las variables de los entornos, cada grupo de variables de un entorno queda guardado en un archivo *properties*. Estos archivos son luego usados por el *Hudson* para conectarse a los diferentes entornos.

En caso de seleccionar un entorno en concreto accederemos a otra ventana donde se puedan modificar las propiedades de este entorno y persistirlas en base de datos.

ALIAS ENTORNO	EQUIPO	RUTA BATCH	BATCH PATH	URL FLEX	FTP PATH	SERVER	SERVER INSTANCE	est.	SERVER INSTANCE	est.	SERVER INSTANCE	est.
REN1	EVERIS	rgpsapl4	rendimiento_1	dgpsweb1	/rendimiento	rgpsapl1	weblogic-1		weblogic-admin		weblogic-ren	

Ilustración 2.23: Campos de la pantalla de información de entornos

Como se ve en la imagen anterior estos van a ser los campos que se muestran en la pantalla.

2.7.3. Edición documento entornos

En la pantalla ‘edición’ se muestran los parámetros guardados en la base de datos. Estos se leen de base de datos y se imprimen por pantalla. En esta pantalla, se permite editar cualquiera de los datos de entorno y hacer un *commit* en base de datos, de manera que se pueda actualizar los datos de los entornos.

Ilustración 2.24: Lista datos de un entorno

Así pues quedaría de esta manera la lista de datos de cada entorno (ilustración 2.24).

2.7.4. Aplicativo externo .jar

Es un aplicativo externo que carece de capa de presentación y tiene un único servicio que se encarga de persistir la información en la base de datos. Utilizando la funcionalidad del DAO (*Data Access Object*) persiste los datos en un modelo *(@Entity*) del aplicativo principal.

2.8. Documento técnico de la aplicación

En este apartado nos podremos encontrar la funcionalidad interna de la aplicación y los servicios que incorpora.

2.8.1. Pantallas del aplicativo

La aplicación usa *Maven* para la gestión del desarrollo del software. El proyecto dispondrá de cuatro sub-proyectos. Cada uno de los módulos representa una de las capas en el patrón de arquitectura Modelo Vista Controlador, más un servicio adicional.

La capa de modelo es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a la base de datos ya sea para persistir información o para extraer información. También es la encargada de gestionar la estructura que va a tener la información que ahí se guarda, es decir, la estructura de las entidades. Por lo tanto, la aplicación va a disponer de dos tipos de clases java, las encargadas de definir la estructura (entidades) y las encargadas de realizar las acciones a la base de datos (DAOs).

El segundo componente de MERT es la capa de servicios o como se ha llamado anteriormente controlador, es el encargado de responder a los eventos (normalmente por acciones del usuario) e invoca métodos de la parte de persistencia para interaccionar con los datos almacenados. También se encarga de devolver información a la capa Web. Se puede entender el controlador como un intermediario entre la parte Web/usuario y la base de datos. En esta parte de la aplicación se dispone de dos tipos de clases java. En primer lugar se tiene los DTO (*Data Transfer Object*) que serán los encargados de conectar las capas de servicio y la de vista, se usan como contenedores para transferir la información. También se dispone de los propios servicios que realizaran acciones como rellenar la información de los DTO, conectarse al *Weblogic* vía un cliente JMX, acceder a los DAOs (*Data Access Object*) de la base de datos, conectarse vía FTP al frontal de un entorno, conectarse vía SSH a la parte *Batch*, conectarse al *Hudson* para lanzar la tarea de entornos o modificar el archivo de ‘enviroment.txt’. También permite guardarlo en la base de datos.

En tercer lugar se dispone de la parte web. Esta capa está destinada a la interacción del usuario con la aplicación, muestra la información de la parte de base de datos y de la capa de servicios de manera que el usuario final lo entienda y pueda interaccionar con ella. Este sub-proyecto incorpora los ficheros *.jsp*, las imágenes y los documentos para la representación visual (*.css)* y, por último, las clases que responden a las peticiones del cliente.

Para entenderlo mejor, utilizando el protocolo HTTP se lanza una petición desde el cliente (*request*) y el servidor devuelve una respuesta (response). La respuesta contiene el estado de la información sobre la petición y puede contener el contenido solicitado.

Mediante los archivos 'pom.xml' se configura la estructura de la aplicación, que son los archivos de configuración de *Maven*. Tenemos 5 'pom.xml' en la aplicación MERT: el POM padre y el de cada uno de sus sub-módulos. El 'pom.xml' genérico dispone de las siguientes configuraciones:

- GroupId: everis.orchestrator.mert-sps
- ArtifactId: mert-sps-war
- Packaging: pom
- Version:0.0.1-SNAPSHOT
- DependencyManagement: (librerías necesarias para el proyecto).
- Modules
    - Mert-sps-persistence
    - Mert-sps-services
    - Mert-sps-aux_service
    - Mert-sps-web

.

El *groupId* es una etiqueta en forma de empaquetado que identifica de manera única el proyecto. El *artifactId* es el nombre que va a recibir el artefacto de *Maven* una vez creado. El *packaging* es el tipo de empaquetado, en este caso es de tipo POM (Project Object Model) porque actúa como un contenedor de sub-módulos. En el siguiente campo se marca el nombre de la versión y por último, las dependencias que se van a utilizar en los *POM* hijos, de esta manera el control de versiones está centralizado. También se indica los módulos que van a componer el POM.

El pom.xml del sub-proyecto de persistencia contiene los siguientes parámetros:

1. *GroupId*: everis.orchestrator.mert-sps
2. *ArtifactId*: mert-sps-persistence
3. *Packaging*: *jar*
4. Version:0.0.1-SNAPSHOT
5. *Parent*:
    - *GroupId*: everis.orchestrator.mert-sps

- *ArtifactId*: mert-sps-war
- *Version*:0.0.1-SNAPSHOT

6. *Dependencies*: (librerías necesarias para el proyecto).

A diferencia del POM padre el tipo de empaquetado es de tipo *jar*. Para construir la estructura es necesario determinar el POM padre, como se ha añadido en los parámetros. En último lugar se añaden las librerías necesarias para la parte de persistencia. Librerías de *MySQL*, *Hibernate* y *Jpa* (gestión de la base de datos), de Spring (para conectar las diferentes capas de la aplicación), de la base de datos H2 y de las de JUnit y Log4java.

El POM de la capa de servicios contiene los siguientes *tags*:

1. *GroupId*: everis.orchestrator.mert-sps
2. *ArtifactId*: mert-sps-services
3. *Packaging*: *jar*
4. *Version*:0.0.1-SNAPSHOT
5. *Parent*:
   - *GroupId*: everis.orchestrator.mert-sps
   - *ArtifactId*: mert-sps-war
   - *Version*:0.0.1-SNAPSHOT
6. *Dependencies*: (librerías necesarias para el proyecto).

La diferencia con el de persistencia reside en que se descarga el módulo de persistencia. También se descarga las librerías de *Spring*, *Junit* y *Log4java*, como en el caso de persistencia. Como librerías nuevas se descarga Wlfullclient (Weblogic), *Commons-net* (FTP), *ml-ant-http* (REST) y JSCH (SSH).

En el caso del servicio adicional lo único que varía (respecto al POM de servicios) es el nombre del *ArtifactId*. Este servicio es el encargado de guardar en base de datos los parámetros de los entornos.

En el último POM de configuración tenemos los siguientes parámetros:

1. *GroupId*: everis.orchestrator.mert-sps
2. *ArtifactId*: mert-sps-web

3. *Packaging*: *war*
4. *Version*:0.0.1-SNAPSHOT
5. *Parent*:
    - *GroupId*: everis.orchestrator.mert-sps
    - *ArtifactId*: mert-sps-war
    - *Version*:0.0.1-SNAPSHOT
6. *Dependencies*: (librerías necesarias para el proyecto).

Para este POM a diferencia del de la capa de servicios cambia el tipo de empaquetado y en vez de descargarse como dependencia el módulo de persistencia se descarga el módulo de servicios.

Como librerías a parte de las de Spring y Junit también se descarga la servlet-api, la JSTL y la Taglibs.

En la capa de persistencia se tiene 5 clases. Una de ellas es la entidad en la base de datos. La encargada de definir que campos va a tener la BBDD y la organización de los mismos. Por otro lado se dispone de un DAO concreto y otro genérico. El DAO concreto va a extender del genérico.

Dentro de la carpeta de recursos se tiene el archivo de configuración de *Spring* y el archivo de configuración de la base de datos.

En la capa de servicios o *business* se dispone de diferentes archivos DTO encargados de recoger los datos de los diferentes *Beans* (de la capa de servicio) para poder enviarlos a la capa de presentación.

Se dispone de una clase de variables globales para poder enviar información de una capa a la otra. También los servicios:

- Conexión al Weblogic (lectura versiones)
- Conexión SSH al Batch (lectura versiones y bindings)
- Conexión FTP al frontal (lectura versiones)
- Conexión SSH a la máquina del Weblogic (lectura bindings)
- Clase de Threads para la pantalla principal
- Clase de Threads para la segunda pantalla
- Lectura entorno base de datos
- Modificación base de datos

- Lectura instancia
- Generar txt
- Ejecutar Job del Hudson

Por ultimo en la carpeta de *resources* se tiene el archivo de configuración de *Spring*.

En última instancia se encuentra el controlador encargado de dar respuesta a las peticiones web. Se tiene 3 archivos *.jsp* uno por cada pantalla, más un archivo CSS y las imágenes requeridas.

Se tiene el archivo de configuración de *Spring* más el archivo ‘web.xml’.

3. Desarrollo de la aplicación

En el tercer capítulo podemos encontrar los temas relacionados con el desarrollo de la aplicación. En primer lugar encontramos la arquitectura y estructura de la aplicación, el esqueleto del aplicativo. En segundo lugar podemos ver la guía de navegación de MERT, el escenario que se va encontrar el usuario. Posteriormente, encontramos el cuerpo de la aplicación, esto incluye las funciones que se implementan, servicios, datos persistidos en BBDD, … por otra parte, también encontramos la tarea del *Hudson* que ejecuta el aplicativo y el *target* de *Ant* que llama esta tarea para generar archivos *properties* necesarios para la ejecución de tareas del proyecto Zeus. En último lugar encontramos los requerimientos de la instalación.

3.1. Arquitectura y estructura de la aplicación

El proyecto se ha desarrollado como un proyecto Web basado en *Apache Maven*. Se ha creado un proyecto padre formado por 4 sub-proyectos hijo y un archivo de configuración ‘pom.xml’ (ver ilustración 3.1).

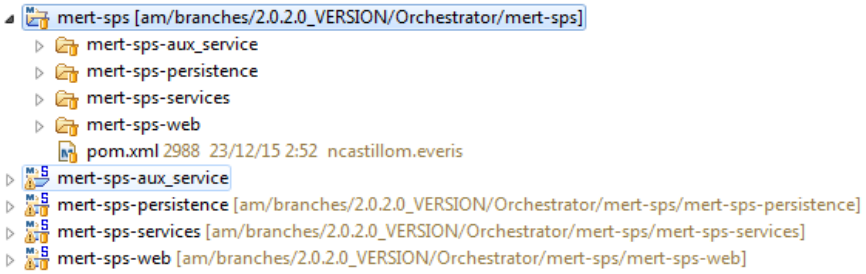


Ilustración 3.1: Estructura de la aplicacion en el SVN

El archivo ‘pom.xml’ contiene, por un lado, el *package* del padre, el nombre del artefacto, el tipo de empaquetado (POM), y el nombre de la versión. Por otro lado, el archivo contiene las dependencias (librerías necesarias) y los módulos de los que se compone. Las dependencias están definidas dentro del *dependencyManagement*. De esta manera, si se ha de cambiar la versión de una dependencia se puede hacer mediante la modificación de una línea y no de cada archivo ‘pom.xml’ que la contenga.

```
176    <modules>
177        <module>mert-sps-aux_service</module>
178        <module>mert-sps-persistence</module>
179        <module>mert-sps-services</module>
180        <module>mert-sps-web</module>
181    </modules>
```

Ilustración 3.2: Manera en que el archivo POM del padre define a sus módulos hijo que lo componen

mert-sps/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>everis.orchestrator.mert-sps</groupId>
    <artifactId>mert-sps-war</artifactId>
    <packaging>pom</packaging>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <java-version>1.8</java-version>
        <org.springframework-version>4.0.3.RELEASE</org.springframework-version>
        <org.hibernate-version>4.3.10.Final</org.hibernate-version>
        <spring.security.version>3.2.5.RELEASE</spring.security.version>
    </properties>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
                <version>${org.springframework-version}</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-web</artifactId>
                <version>${org.springframework-version}</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
```

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Ilustración 3.3: Archivo *pom.xml* del proyecto padre

Por otro lado, se encuentran los demás sub-proyectos que contienen las clases java, archivos de configuración XML (véase ilustración 3.2). Cada sub-proyecto contiene un archivo *pom.xml* donde indica el proyecto padre, el tipo de empaquetado y las dependencias, dentro de estas dependencias se incluye también el sub-modulo que corresponde (ejemplo archivo *pom.xml* en la ilustración 3.3). Es decir el *pom.xml* de la parte Web contiene como dependencia el sub-proyecto mert-sps-services, el cual contiene a su vez el sub-módulo mert-sps-persistence. De esta manera cuando se hace un *mvn default* (*mvn install*) el archivo *.war* final contiene los demás sub-proyectos dentro de sus librerías y puede acceder a sus clases.

Existen otros tipos de archivos *.xml* que contienen la configuración de *Spring* (véase el anexo 6.2) y la configuración de *Hibernate* (aunque esta delegada a la configuración del *framework* de *Spring*). Junto con el archivo web.xml, conforman el sitio Web.

Como se aprecia en la ilustración 3.4. El frontal (*View*) recibe la petición la cual pasa a la capa de servicios en caso de necesitarlo. En último lugar dependiendo del servicio al que se ha llamado leer, escribe, o no accede a base de datos y finalmente, devuelve la información al frontal.

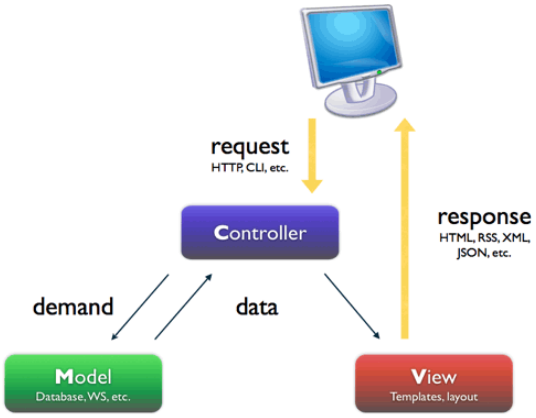


Ilustración 3.4: Petición a un modelo MVC.

La aplicación se ha diseñado en base a la arquitectura Modelo-Vista-Controlador (MVC), tal y como lo muestra la ilustración 3.5.

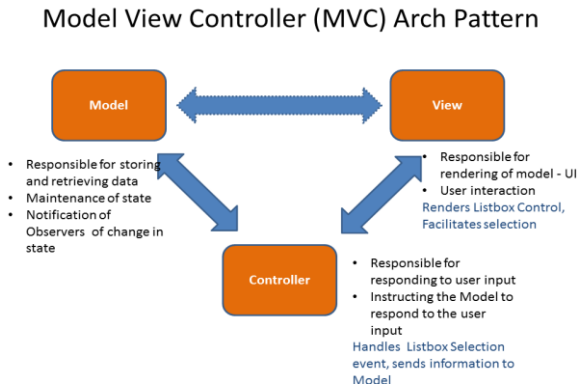


Ilustración 3.5: Modelo MVC

El patrón de arquitectura de software MVC es una estructura que propone separar el código de los programas por sus diferentes responsabilidades.

El fundamento es separar el código en tres capas:

- La capa de modelo o persistencia: Es en la que se trabaja con la base de datos, por lo tanto, se encarga de acceder a la información y generar las entidades con las que

se trabaja. Normalmente se trabaja con las clases CRUD que tienen como métodos crear (*Create*), leer (*Read*), actualizar (*Update*) y eliminar (*Delete*) los datos.

- La capa de vista o Web: Es la capa de presentación, la parte de la aplicación con la que el usuario final va a interaccionar. En la vista tenemos los códigos HTML que nos permite mostrar la salida.
- La capa de controlador o de servicio: Contiene el código que responde a las peticiones del usuario e interacciona si es necesario con la capa de persistencia.

De esta manera cada sub-proyecto es una capa del modelo MVC. Lo forman la capa de persistencia, la capa de servicios y la capa Web, más un servicio adicional. Cabe destacar que no se utilizan archivos *.html* sino que se utilizan JSP (*Java Server Pages*). Es una tecnología que permite crear páginas Web con programación en Java. Permite recuperar los datos de la capa de servicios e imprimirlos por pantalla de manera dinámica.

A continuación se muestra todas las clases que conforman el aplicativo MERT.

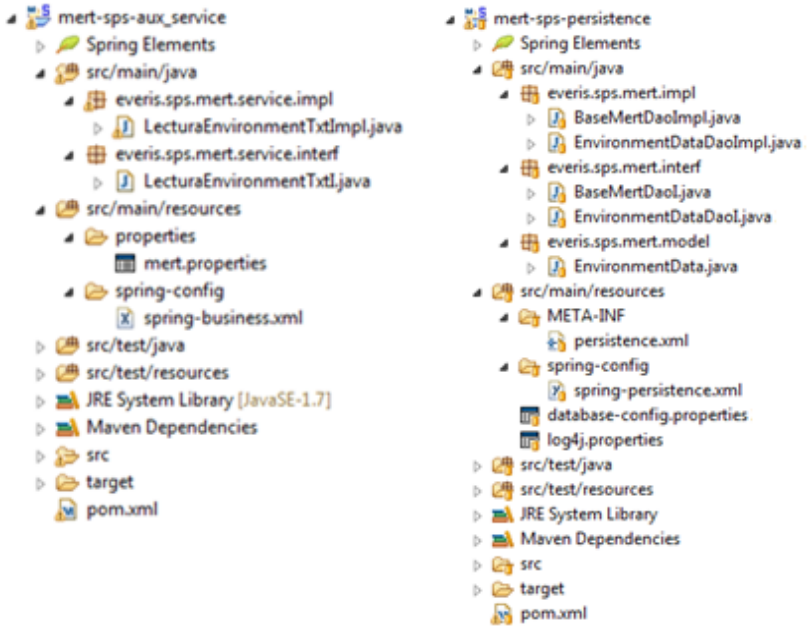


Ilustración 3.6: Los sub-módulos de aux_service (izquierda) y de la capa de persistencia (derecha)

Como se aprecia en la anterior ilustración (3.6) se dispone del servicio adicional que contiene una única clase y su interfaz. Esta clase es añade en el contexto de *Spring* gracias a la anotación *@Service*.

Por otro lado, se tiene la parte del proyecto que controla la persistencia en base de datos. La estructura de una entidad dentro de la base de datos se define mediante *@Entity* (de esta manera se carga su esqueleto). También se dispone de los métodos DAO que son los encargados de hacer las acciones transaccionales a la base de datos. Estas clases se anotan mediante *@Repository*, lo cual añade en el contexto de Spring el *Bean* 'x' perteneciente a la capa de base de datos. Las otras anotaciones que se usan son *@PersistenceContext* que sirve para crear una instancia de la *EntityManagerFactory* la cual se utiliza para interactuar con la base de datos. Por otro lado, cada método se define con '@Transactional' para que cada vez que se efectúe una acción se produzca la transacción.

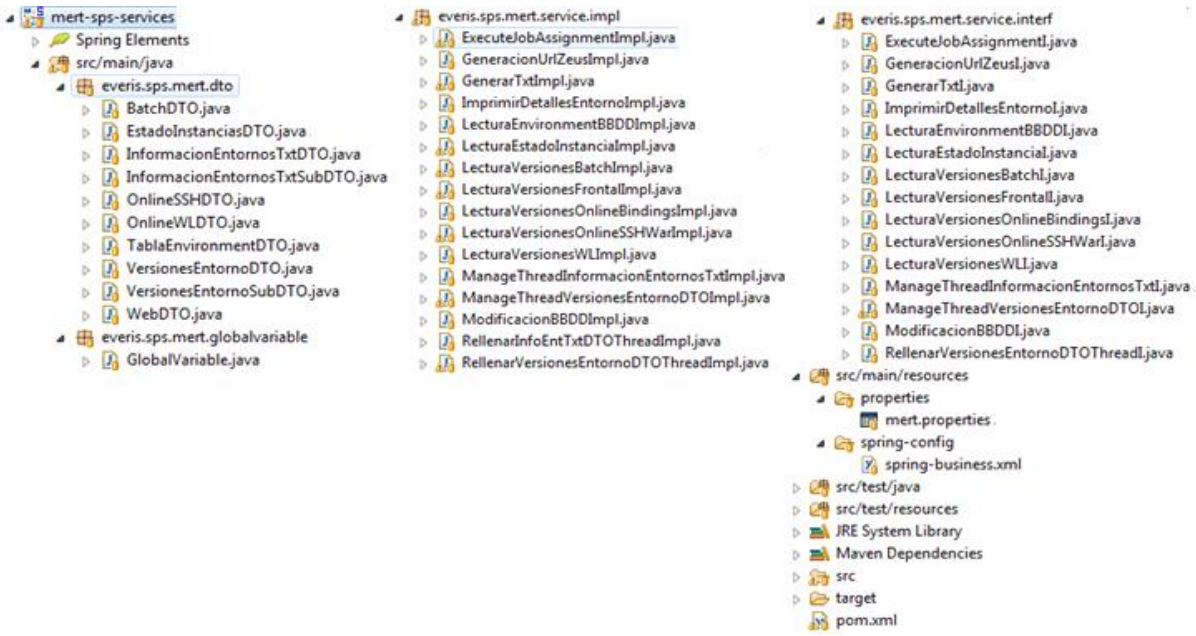


Ilustración 3.7: El sub-proyecto de la capa de servicios

En la capa de servicios hay 2 tipos de clases los DTO y los *Beans* (véase la ilustración 3.7). Los DTO son clases sin ningún tipo de configuración específico formado por un conjunto de atributos (con sus *getters* y *setters*). Son usados para la comunicación entre capas. Es decir, una clase de la capa Web llama a un servicio el cual le devolverá un DTO que se utiliza por una página JSP para dibujar los datos dinámicamente.

Los *Beans* que son clases que se inyectan en el contexto de Spring. Se identifican mediante la anotación *@Service*. Son las encargadas de la lógica de la página.

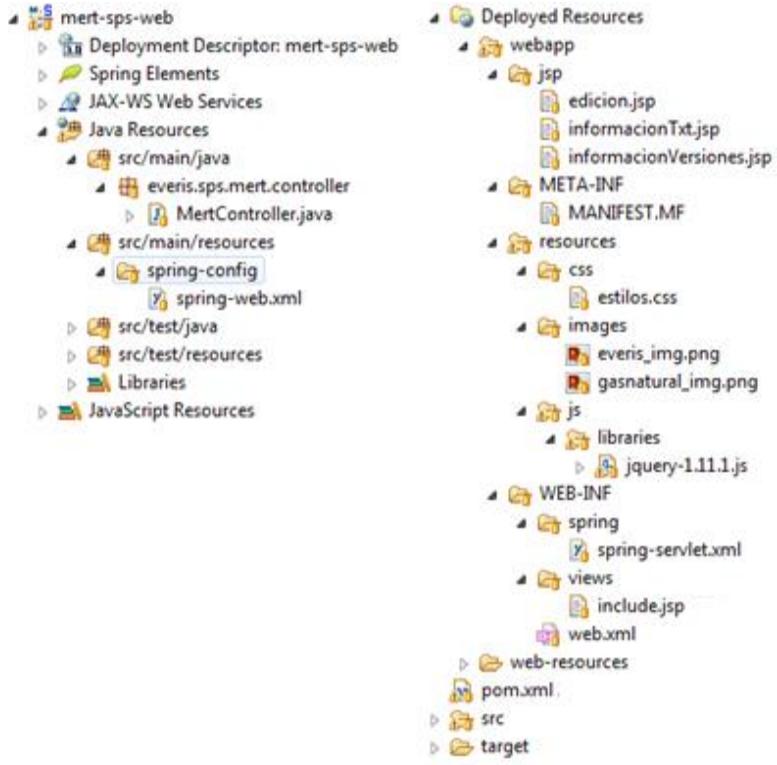


Ilustración 3.8: Sub-módulo de la capa Web

En último lugar, se dispone de la capa Web (ilustración 3.8) donde se encuentra la clase cuya función es responder a las peticiones HTTP. Esta clase se inyecta al contexto de *Spring* utilizando la anotación *@Controller*. La cual llama a los servicios necesarios y carga la página JSP pertinente.

Contexto de Spring

La ilustración 3.9 muestra el archivo de configuración de *Hibernate* para trabajar con la base de datos, no contiene ningún parámetro porque Spring permite delegar toda esa lógica a su 'persistence-context.xml' el cual contiene todos los datos relevantes de la base de datos

(usuario, contraseña, dialecto, URL,…). Se puede apreciar en la ilustración 3.10, todos los datos se leen de un archivo *properties* para una gestión más adecuada.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
    <persistence-unit name="persistence-unit-mert" >
    </persistence-unit>
</persistence>
```

Ilustración 3.9: Archivo de configuración de *Hibernate*.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans  xmlns="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd">

    <context:annotation-config/>
    <context:component-scan base-package="everis.sps.mert.*"></context:component-scan>


    <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location">
            <value>classpath:/database-config.properties</value>
        </property>
    </bean>

    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>


    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory" />
    </bean>

    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="persistenceXmlLocation" value="classpath:/META-INF/persistence.xml" />
        <property name="persistenceUnitName" value="persistence-unit-mert" />
        <property name="dataSource" ref="dataSource" />
        <property name="jpaVendorAdapter">
            <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
                <property name="showSql" value="${hibernate.show_sql}" />
                <property name="generateDdl" value="${jpa.generateDdl}" />
                <property name="databasePlatform" value="${hibernate.dialect}" />
            </bean>
        </property>
        <property name="packagesToScan" value="**" />
        <property name="jpaProperties" ref="hibernateProperties" />
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager" />

    <bean id="hibernateProperties" class="org.springframework.beans.factory.config.PropertiesFactoryBean">
        <property name="properties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
                <prop key="hibernate.generate_statistics">${hibernate.generate_statistics}</prop>
            </props>
        </property>
    </bean>
</beans>
```

Ilustración 3.10: Contexto de *Spring* de la capa de persistencia

La aplicación esta diseñada para que el archivo ‘web.xml’ cargue el contexto de *Spring* de la parte Web y este haga una cadena que recorre la capa Web, pasando por servicios y llegando a persistencia (cada uno carga la configuracion de su capa contigua). Asi pues en el archivo definitivo estarian todos los tags incluidos en cada uno de los contextos de *Spring* mas, en tiempo de ejecucion, los *Beans* de cada una de las capas.

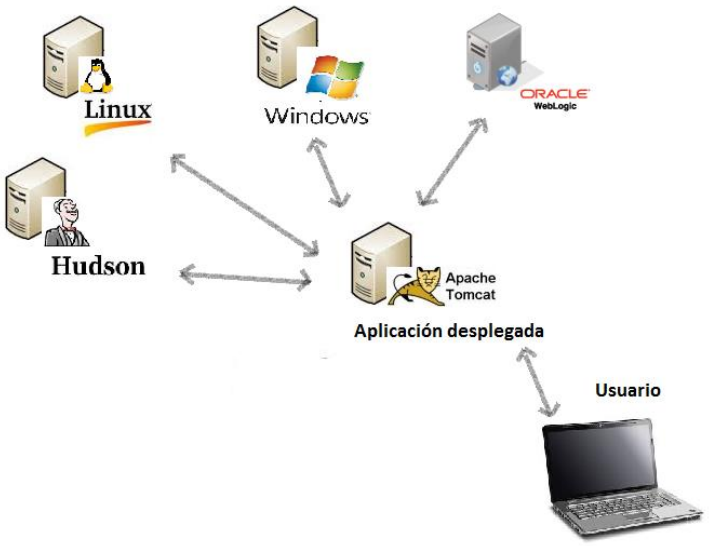


Ilustración 3.11: Comunicación del aplicativo con las diferentes maquinas

Como se muestra en la ilustración 3.11, estas son las comunicaciones que va a establecer el aplicativo. Para dar respuesta a estas peticiones se ha definido un modelo cliente servidor.

En esta figura se aprecia que el nodo central es el servidor *Apache Tomcat* donde está desplegada la aplicación. Es el encargado de hacer todas las peticiones HTTP. Según la petición del usuario el aplicativo decide que conexiones necesita establecer.

Excepto la conexión con el servidor de aplicaciones donde está el *Hudson*, el resto se repiten tantas veces como entornos hay en el proyecto Zeus.

3.2. Guía de navegación web

La navegación entre pantallas se hace siguiendo las flechas que se indican en el diagrama 3.12. La pantalla principal es la de información de las versiones, en esta pantalla se permite conectar con el frontal de un entorno para testear la aplicación allí alojada. Permite filtrar según los contenidos que aparecen en los datos de un entorno. También permite ampliar la información de un entorno en concreto.

Ilustración 3.12: Diagrama de flujo del aplicativo

En la segunda pantalla se muestra la información de los entornos, referente a usuarios de máquinas entornos, etc. esta página también posee filtro. Desde esta vista se puede ejecutar el *job* del *Hudson* o generar el archivo *txt* según lo que hay guardado en base de datos.

Al presionar en un entorno en concreto se accede a la página de edición donde se puede alterar los datos persistidos en base de datos.

3.3. Implementación de la aplicación

A continuación se describe cada una de las conexiones requeridas en la aplicación y el funcionamiento interno de la aplicación MERT.

3.3.1. Información versiones

Parte HTML

La página del menú principal (ilustración 3.13) se ha creado con JSP, para generar las paginas HTML resultantes. Para añadir funcionalidad a las paginas JSP se ha utiliza la librería JSTL (*JSP Standard Tag Library*) para añadir dinamismo a la página. Los datos que se muestran por pantalla se obtienen a partir de un DTO (*Data Transfer Object*). Básicamente es una clase con diversos atributos y con métodos que permiten escribir y leer, los valores de estos atributos (métodos *getters* y *setters*).

Ilustración 3.13: Página de inicio, muestra las versiones por cada entorno

La información que a priori se muestra no es el total de información que hay en la página HTML. La información ampliada que vemos en la ilustración 3.14 esta oculta, en caso de presionar encima de un entorno se muestra su información detallada en forma de *popup*. Se añade una capa entre el *popup* y la pantalla para que no se pueda hacer ninguna petición más mientras el *popup* esté abierto.

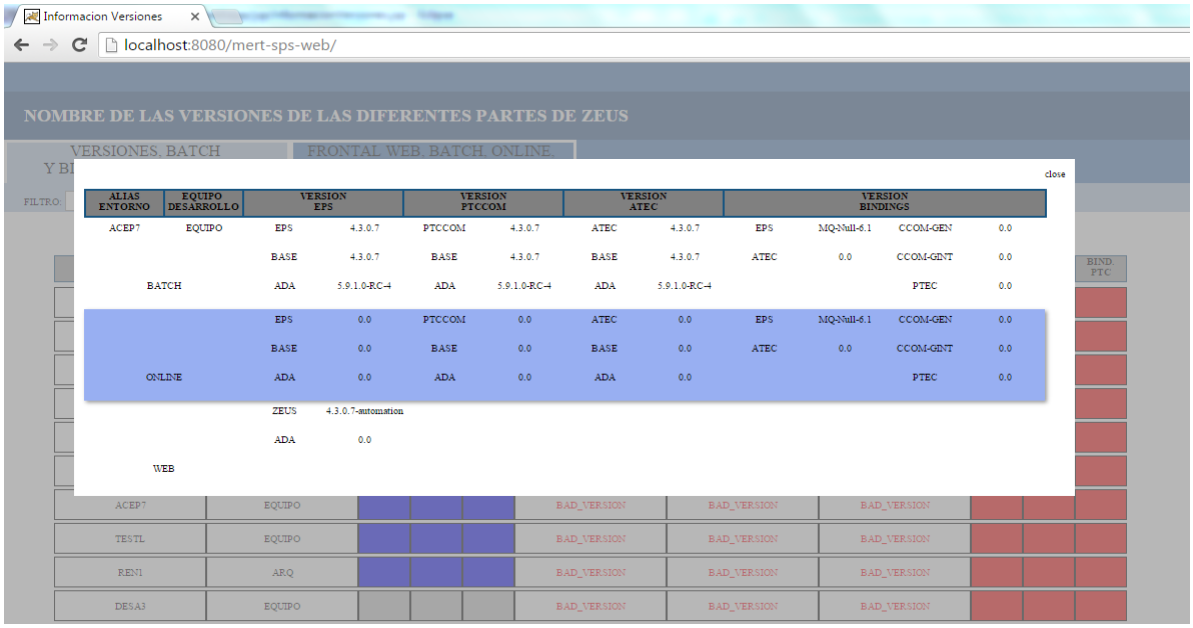


Ilustración 3.14: *Popup* de la página de inicio con la información completa de un entorno

Por otra parte tenemos el filtro para encontrar un entorno introduciendo su alias por ejemplo. Por cada botón que se levanta (*onkeyup*) se activa el filtro y busca coincidencias en

los parámetros de texto del documento. Oculta los entornos que no cumplen con los parámetros que se han introducido.

Ilustración 3.15: Pagina inicial a la cual se le ha aplicado un filtro 'TEST1'

Si un entorno tiene un subsistema desplegado se muestra de color azul, en caso negativo de color gris. Por otro lado en el caso de los *Bindings* hay 3 colores: color verde, si coinciden entre las diferentes máquinas de un entorno y están activos; color rojo si no coinciden; y color gris si coinciden pero no están activos.

En el apartado de versiones de Zeus en el caso de no coincidir las versiones entre las diferentes partes aparece BAD_VERSION en color rojo. En caso contrario aparece el nombre de la versión.

Lógica de la pagina

Al acceder a la página de inicio se llama a un servicio que se llama 'ManageThreadVersionesEntornoDTOI' que es el encargado de crear 'n' hijos. Con 'n' el total de entornos disponibles. Las clases hijo llaman al servicio 'RellenarVersionesEntornoDTOThreadImpl' que es la encargada de llevar a cabo las consultas a las diferentes máquinas que componen un entorno:

En primer lugar se genera la URL del frontal Zeus al entorno.

En segundo lugar se conecta mediante una conexión SSH al servidor con sistema operativo Linux que contiene la capa *Batch* de la aplicación (en la imagen 3.16 se muestra la librería que se requiere incluir en el archivo pom.xml).

```
79        <dependency>
80            <groupId>com.jcraft</groupId>
81            <artifactId>jsch</artifactId>
82        </dependency>
```

Ilustración 3.16: Dependencia JSCH para conectarse vía SSH

Luego, para conectarse a la maquina remota se ha de introducir el host remoto, el usuario la contraseña y el puerto.

```
JSch jsch = new JSch();
Session session = jsch.getSession(user, host, 22);
session.setPassword(password);
session.setDaemonThread(false);
session.setConfig("StrictHostKeyChecking", "no");
session.setTimeout(30000);//MILISECONDS
session.connect();
Channel channel=session.openChannel("sftp");
channel.connect();
channel.disconnect();
session.disconnect();
```

Ilustración 3.17: Código implementado para una conexión SSH

Mediante las líneas de la imagen 3.17 de código java se ha establecido conexión. Y desconexión. Una vez dentro de la maquina remota se extrae la información de las versiones de las cabeceras de diferentes archivos desplegados en esta máquina. En el caso de los *Bindings*, se ha accedido a un directorio oculto *.bindings* donde esta guardada la información. En las máquinas que está instalado el servidor *Weblogic* los *Bindings* también se han extraído mediante una conexión SSH.

En tercer lugar se ha conectado a la maquina Windows donde está desplegado el frontal de la aplicación Zeus de un determinado entorno. A estas máquinas se ha establecido una conexión FTP. Para este tipo de conexión se ha requerido de la siguiente librería.

```
<dependency>
    <groupId>commons-net</groupId>
    <artifactId>commons-net</artifactId>
</dependency>
```

Ilustración 3.18: Dependencia commons-net utilizado para la conexión FTP

En la ilustración 3.19 se muestra la implementación de una conexión utilizando el protocolo de transmisión de ficheros (FTP).

```
FTPClient ftp = new FTPClient();
ftp.setConnectTimeout(30000);
ftp.connect(host, 21);
ftp.enterLocalPassiveMode();
ftp.setFileType(FTPClient.BINARY_FILE_TYPE);
ftp.setRemoteVerificationEnabled(false);
ftp.login(user, password);
ftp.disconnect();
```

Ilustración 3.19: Implementación de la conexión vía FTP

Como se explica en el punto 2.3.2., en el estudio previo se utiliza una conexión en modo pasivo.

En último lugar utiliza un servicio que se conecta mediante un conector JMX a los *MBeans* para extraer la información de cada aplicación desplegada, mediante la lectura de atributos. En la ilustración 3.20 se muestra la implementación de una conexión JMX.

```
String protocol = "t3";
Integer portInteger = Integer.valueOf(portString);
int port = portInteger.intValue();
String jndiroot = "/jndi/";
String mserver = "weblogic.management.mbeanservers.domainruntime";
JMXServiceURL serviceURL = new JMXServiceURL(protocol, hostname, port,jndiroot + mserver);
Hashtable h = new Hashtable();
h.put(Context.SECURITY_PRINCIPAL, username);
h.put(Context.SECURITY_CREDENTIALS, password);

h.put(JMXConnectorFactory.PROTOCOL_PROVIDER_PACKAGES,"weblogic.management.remote");
connector = connectWithTimeout(serviceURL, 50000, TimeUnit.MILLISECONDS, h);
connection = connector.getMBeanServerConnection();
connector.close();
```

Ilustración 3.20: Implementación mediante la librería 'wlfullclient.jar' de una conexión JMX al Weblogic

Una vez los procesos hijos acaban, y guardan la información en una variable global que contiene diferentes DTO, el proceso padre continua con la ejecución de la página JSP.

Todos los datos referentes a direcciones URL, usuario, *password*, etc. se extraen de la base de datos donde por cada entorno se tiene todos los datos relevantes.

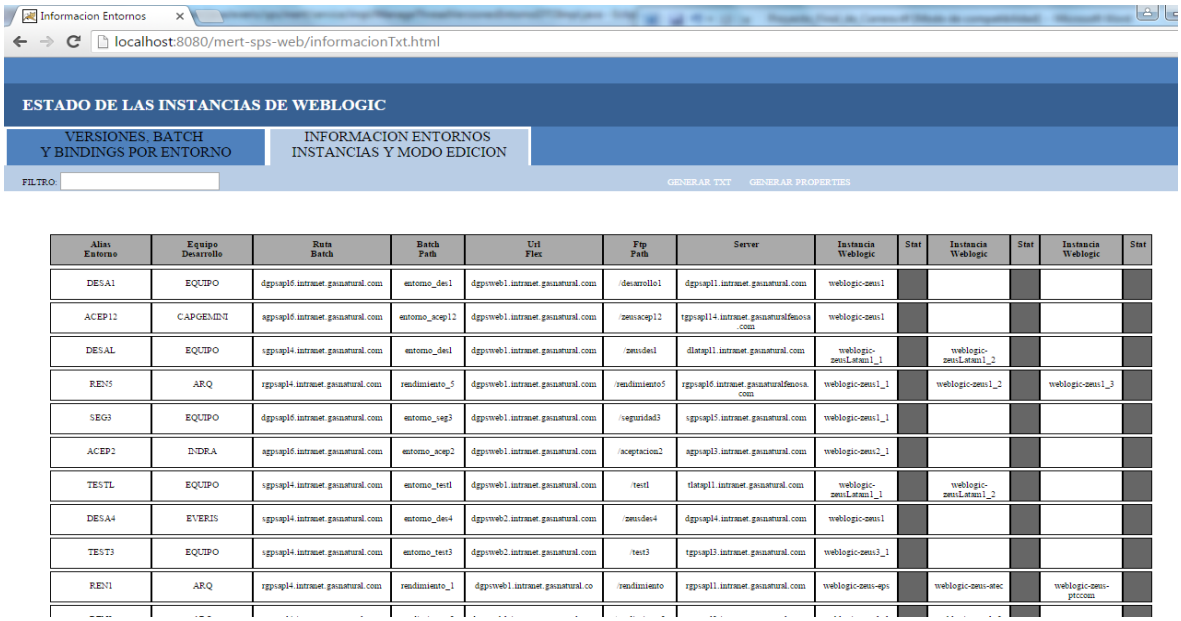
3.3.2. Información entornos y estado instancias

Parte HTML

Como la página de inicio, la página de estado de las instancias de *Weblogic*, también está hecha a partir de JSP. Se extrae información de los entornos de un DTO que ha proporcionado la capa de servicios. Cada fila representa un entorno.

Esta página permite también el filtrado que se ha visto en el punto anterior, para facilitar al usuario la búsqueda de un entorno.

En la imagen 3.21 se muestra la visualización de la segunda pantalla.



Alias Entorno	Equipo Desarrollo	Ruta Batch	Batch Path	Url Flex	Ftp Path	Server	Instancia Weblogic	Stat	Instancia Weblogic	Stat	Instancia Weblogic	Stat
DESA1	EQUIPO	dgpsapl6.intranet.gasnatural.com	entorno_des1	dgpsweb1.intranet.gasnatural.com	/desarrollo1	dgpsapl1.intranet.gasnatural.com	weblogic-zeus1					
ACEP12	CAPGEMINI	agpsapl6.intranet.gasnatural.com	entorno_acep12	dgpsweb1.intranet.gasnatural.com	/zeusacep12	tgpsapl14.intranet.gasnaturalfenosa.com	weblogic-zeus1					
DESAL	EQUIPO	sgpsapl4.intranet.gasnatural.com	entorno_des1	dgpsweb1.intranet.gasnatural.com	/zeusdes1	dlatapl1.intranet.gasnatural.com	weblogic-zeusLatam1_1		weblogic-zeusLatam1_2			
REN5	ARQ	rgpsapl4.intranet.gasnatural.com	rendimiento_5	dgpsweb1.intranet.gasnatural.com	/rendimiento5	rgpsapl6.intranet.gasnaturalfenosa.com	weblogic-zeus1_1		weblogic-zeus1_2		weblogic-zeus1_3	
SEG3	EQUIPO	dgpsapl6.intranet.gasnatural.com	entorno_seg3	dgpsweb1.intranet.gasnatural.com	/seguridad3	sgpsapl5.intranet.gasnatural.com	weblogic-zeus1_1					
ACEP2	INDRA	agpsapl6.intranet.gasnatural.com	entorno_acep2	dgpsweb1.intranet.gasnatural.com	/aceptacion2	agpsapl3.intranet.gasnatural.com	weblogic-zeus2_1					
TESTL	EQUIPO	sgpsapl4.intranet.gasnatural.com	entorno_test1	dgpsweb1.intranet.gasnatural.com	/test1	tlatapl1.intranet.gasnatural.com	weblogic-zeusLatam1_1		weblogic-zeusLatam1_2			
DESA4	EVERIS	sgpsapl4.intranet.gasnatural.com	entorno_des4	dgpsweb2.intranet.gasnatural.com	/zeusdes4	dgpsapl4.intranet.gasnatural.com	weblogic-zeus1					
TEST3	EQUIPO	sgpsapl4.intranet.gasnatural.com	entorno_test3	dgpsweb2.intranet.gasnatural.com	/test3	tgpsapl3.intranet.gasnatural.com	weblogic-zeus3_1					
REN1	ARQ	rgpsapl4.intranet.gasnatural.com	rendimiento_1	dgpsweb1.intranet.gasnatural.co	/rendimiento	rgpsapl1.intranet.gasnatural.com	weblogic-zeus-eps		weblogic-zeus-srac		weblogic-zeus-ptccom	

Ilustración 3.21: Pagina de las instancias a *Weblogic*

En las últimas 6 casillas de cada entorno se muestra el nombre de la instancia de *Weblogic* y su estado. Normalmente se van a dar dos escenarios: que esté de color verde (nodo funciona), o que esté en gris (error en el nombre de la instancia).

En esta pantalla, en blanco, se dispone de dos botones ‘GENERAR TXT’ el cual sirve para generar el archivo ‘enviroment.txt’ en el *Hudson* y el botón ‘GENERAR PROPERTIES’ que ejecuta una tarea del *Hudson*.

En caso de presionar sobre un entorno en concreto se accede a la página de edición.

Lógica de la página

La pantalla al ejecutarse llama a un servicio de “ManageThreadInformacionEntornosTxtI” que es el encargado de generar ‘n’ *Threads*. Con ‘n’ el número de entornos. Por cada *Thread* hijo se establece una conexión mediante un conector JMX al Weblogic para comprobar el estado de cada una de las instancias.

Una vez recuperados estos datos, se lee de la base de datos todos los datos que se van a mostrar en la pantalla y se pasan a una variable global que es una lista de DTOs que la página JSP va a interpretar e imprimir por pantalla.

En caso de presionar sobre el botón de ‘GENERAR TXT’ se genera un archivo con los datos persistidos en base de datos y genera un archivo ‘environment.txt’ con la información de todos los entornos. Una vez creado establece una conexión SSH y se guarda en una carpeta específica de la máquina donde está desplegado el *Hudson* (ilustración 3.22). Una vez guardado se corta la conexión con la máquina del *Hudson*.

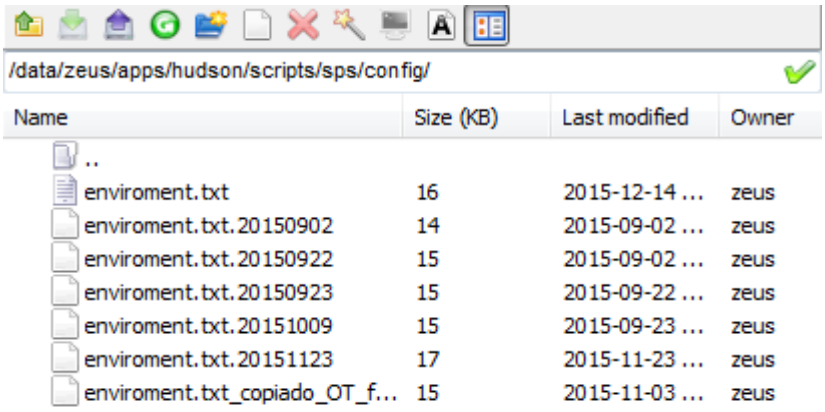


Ilustración 3.22: Directorio donde se guarda el archivo ‘*enviroment.txt*’ que genera MERT

En último lugar, el botón de 'GENERAR PROPERTIES' es el encargado mediante comunicación REST de ejecutar una tarea del *Hudson* que genera los archivos *properties*. Se explica más en detalle en el punto 3.4.

Para la comunicación vía API REST se utiliza la librería:

```
<dependency>
    <groupId>org.missinglink</groupId>
    <artifactId>ml-ant-http</artifactId>
</dependency>
```

Ilustración 3.23: Libreria ml-ant-http utilizado para establecer conexión vía REST al *Hudson*

Como se aprecia en la imagen 3.24, se crea una instancia de la clase 'HttpClientTask()' donde se pasan la URL del *Hudson*, el puerto, las credenciales y el *job* concreto y finalmente, se ejecuta.

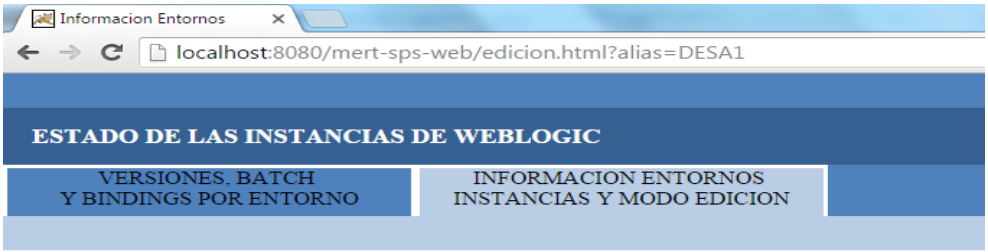
```
1  package everis.sps.mert.service.impl;
2
3  import org.missinglink.ant.task.http.CredentialsNode;
4  import org.missinglink.ant.task.http.HttpClientTask;
5  import org.springframework.stereotype.Service;
6
7  import everis.sps.mert.service.interf.ExecuteJobAssignmentI;
8
9  @Service
10 public class ExecuteJobAssignmentImpl implements ExecuteJobAssignmentI {
11     private HttpClientTask http  = new HttpClientTask();
12     private CredentialsNode credentials = new CredentialsNode();
13     private int expected = 200;
14     private String serverUrl ;
15     private String jobName;
16     public int ejecucionJob(){
17         this.http.setUrl("http://dzeusub2.intranet.gasnatural.com:8080/hudson/job/ZEUS_ARQ_SPS_enviroment_assignment/build");
18         this.credentials.setUsername("90034270");
19         this.credentials.setPassword("casa123");
20         this.http.addConfiguredCredentials(this.credentials);
21         this.http.setFailonunexpected(true);
22         this.http.execute();
23         return expected;
24     }
25 }
26
```

Ilustración 3.24: Clase encargada de lanzar el *job* del *Hudson*

3.3.3. Edición documento entornos

Parte HTML

Esta página permite alterar las variables que están escritas en una línea de base de datos. Al presionar sobre el botón 'SUBMIT' carga la información de la página en base de datos. En la ilustración 3.25 se muestra la pantalla de edición.



alias Entorno	DESA1
equipo	EQUIPO
pgenviromentclient	dgpsweb1.intranet.gasnatural.com
pgenviromentclient2	
pgenviromentclientuser	desarrollo1
pgenviromentclientpassword	xxx
pgenviromentclientpath	/desarrollo1
pgenviromentjee	dgpsapl1.intranet.gasnatural.com
pgenviromentjeewarscpuser	gps01
pgenviromentjeewarscppassword	xxx
pgenviromentjeewluser	zeusd
pgenviromentjeewlpassword	dgpsapl1
pgenviromentjeewlserverName	weblogic-zeus1
pgenviromentjeewlserverName2	
pgenviromentjeewlserverName3	
pgenviromentjeebatch	dgpsapl6.intranet.gasnatural.com
pgenviromentjeebatchscpuser	gps07
pgenviromentjeebatchscppassword	xxx
pgenviromentjeebatchpath	entorno_des1
pgenviromentbbddwuser	DESA1
pgenviromentbbddwpwd	xxx
pgenviromentbbddwjdbc	jdbc:oracle:thin:@zeusd-scan.intran
pgenviromentbbddbuser	DESA1
pgenviromentbbddbpwd	xxx
pgenviromentbbddbjdbc	jdbc:oracle:thin:@zeusd-scan.intran
pgenviromenttype	O

SUBMIT

Ilustración 3.25: Página de edición con los datos del entorno 'DESA1

Lógica de la página

Mediante un GET se pasan el valor del ‘alias’ de un entorno, con esta información se llama a un método que recoge todos los datos de ese entorno y los muestra por pantalla.

Al presionar al botón ‘SUBMIT’ se hace un POST a la misma URL que guarda los valores en base de datos y redirecciona la petición a la página de ‘Estado de las instancias de *Weblogic*.

3.3.4. Aplicativo externo .jar

Es un servicio que coge un archivo ‘enviroment.txt’ donde están todos los datos de los entornos, separados por diferentes elementos ‘|’, ’&’, ’,’, etc. lo analiza y guarda la información en base de datos. Este archivo ha de cumplir ciertas normas con tal de que las variables se lean correctamente.

3.4. Job de Hudson y script de ANT

La tarea del *Hudson* (ilustración 3.26) se descarga en un archivo local una versión del SPS donde están los scripts necesarios de *Ant*. Una vez descargados ejecuta un *target* de *Ant* que está en ‘${WORKSPACE}/sps-tools/scripts/gnf.sps.zeus.assignment.enviroment.xml’. El *target* se llama ‘*sps.assignment*’ y se le pasa como variable la propiedad ‘h.g.project.name=ZEUS’.

A continuación se muestra un fragmento del código *Ant* que ejecuta la tarea (véase la ilustración 3.27). Básicamente analiza el código y lo divide por filas y cada fila está dividida por columnas y sub-columnas delimitadas por símbolos.

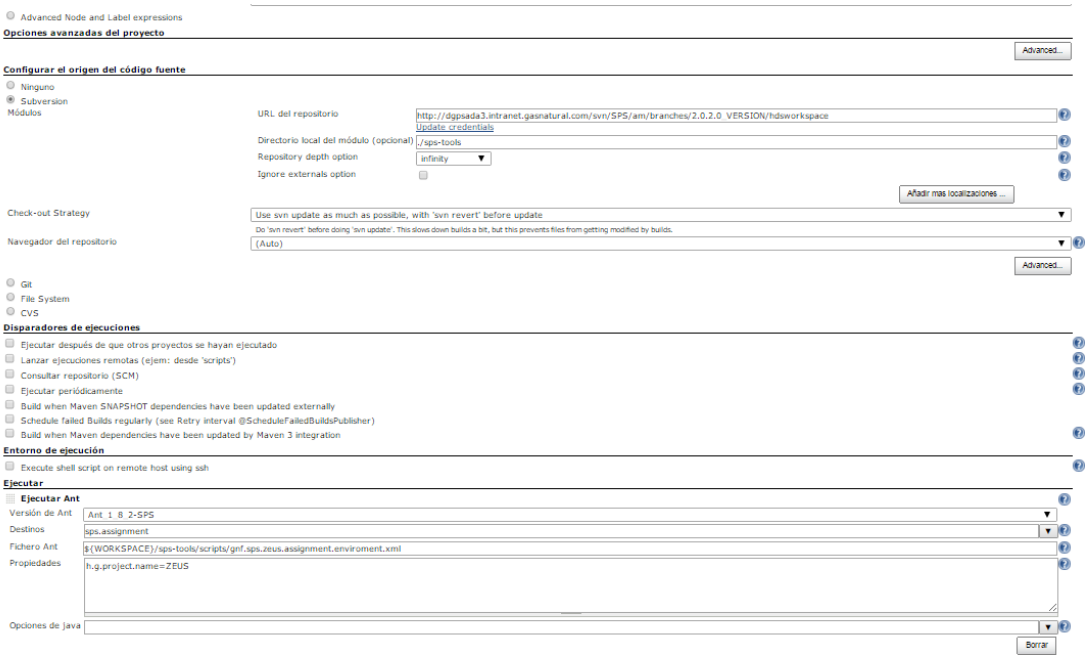


Ilustración 3.26: Configuración del *job* del Hudson

```
17  <target name="sps.assignment" depends="loadProperties">
18
19      <loadfile property="file.enviroment" srcFile="${p.hudson.dir.enviroment}/enviroment.txt"/>
20
21      <echo message="ruta de carga den el servidor ${p.hudson.dir.enviroment}/enviroment.txt" />
22
23
24      <!-- Parseamos el fichero linea a linea -->
25      <for list="${file.enviroment}" delimiter="${line.separator}" param="var.name" parallel="false" >
26          <sequential>
27
28              <!-- Parseamos el fichero columna a columna-->
29
30              <var name="v.l.colum.pos" value="0" />
31              <var name="v.l.var.concat" value=""/>
32              <var name="v.l.var.charter.separator" value="-"/>
33              <var name="v.l.enviroment.fileName" value=""/>
34              <condition  property="v.l.comentcontains">
35                  <contains string="@{var.name}" substring="#ENV"/>
36              </condition>
37              <if>
38                  <equals arg1="${v.l.comentcontains}" arg2="true"/>
39                  <then>
40                      <var name="v.l.comentcontains" unset="true"/>
41                  </then>
42                  <else>
43                      <var name="v.l.comentcontains" unset="true"/>
44
45                      <for list="@{var.name}" delimiter="|" param="var.colValue" parallel="false" trim="true" >
46                          <sequential>
47                              <!-- contador de columna para realizar diferentes acciones-->
48                              <math result="v.l.colum.pos" operand1="${v.l.colum.pos}" operation="+" operand2="1" datatype="int" />
49                              <echo>v.l.colum.pos --- ${v.l.colum.pos}</echo>
50
51                              <var name="v.l.var.concat" value="${v.l.var.concat}@{var.colValue}${v.l.var.charter.separator}"/>
52
53                              <switch value="${v.l.colum.pos}">
54                                  <case value="2">
55                                      <var name="v.l.enviroment.fileName" value="${v.l.var.concat}deploy.properties"/>
56                                      <echo message="v.l.enviroment.fileName ---   ${v.l.enviroment.fileName}"/>
57                                      <echo file="${p.hudson.dir.assignment}/${v.l.enviroment.fileName}" append="false">#=== SPS v1.0.7.0 ===${line.sepa
58                                      <var name="v.l.var.concat" value=""/>
59                                      <var name="v.l.var.charter.separator" value=""/>
60                                  </case>
61                                  <case value="3">
62                                      <var name="v.l.subcolum1.pos" value="0"/>
63                                      <var name="v.l.var.dat1" value=""/>
64                                      <for list="@{var.colValue}" delimiter=";" param="var.str1" parallel="false" trim="true" >
65                                          <sequential>
66                                              <math result="v.l.subcolum1.pos" operand1="${v.l.subcolum1.pos}" operation="+" operand2="1" datatype="int"
67
68                                              <var name="v.l.var.dat1" value="${v.l.var.dat1}@{var.str1}"/>
69                                              <switch value="${v.l.subcolum1.pos}">
70                                                  <case value="1">
71                                                      <echo file="${p.hudson.dir.assignment}/${v.l.enviroment.fileName}" append="true">p.g.enviroment.cl
72                                                      <echo message="p.g.enviroment.client = ${v.l.var.dat1}"/>
73                                                      <var name="v.l.var.dat1" value=""/>
74                                                  </case>
75                                                  <case value="2">
76                                                      <echo file="${p.hudson.dir.assignment}/${v.l.enviroment.fileName}" append="true">p.g.enviroment.cl
77                                                      <echo message="p.g.enviroment.client.user = ${v.l.var.dat1}"/>
78                                                      <var name="v.l.var.dat1" value=""/>
79                                                  </case>
80                                                  <case value="3">
81                                                      <echo file="${p.hudson.dir.assignment}/${v.l.enviroment.fileName}" append="true">p.g.enviroment.cl
82                                                      <echo message="p.g.enviroment.client.password = ${v.l.var.dat1}"/>
83                                                      <var name="v.l.var.dat1" value=""/>
84                                                  </case>
85                                                  <case value="4">
86                                                      <echo file="${p.hudson.dir.assignment}/${v.l.enviroment.fileName}" append="true">p.g.enviroment.cl
87                                                      <echo message="p.g.enviroment.client.path = ${v.l.var.dat1}"/>
88                                                      <var name="v.l.var.dat1" value=""/>
89                                                  </case>
90                                                  <default/>
```

Ilustración 3.27: Script de Ant

3.5. Requerimientos instalación

Al ejecutar el comando *mvn install* (*mvn default*) se genera un paquete *WAR*, que es un comprimido de la aplicación que contiene todas las librerías y módulos de los que está compuesta la aplicación. En la ilustración 3.28 se aprecia cómo está compuesto este paquete. Dentro de la carpeta 'lib' están los demás paquetes de la aplicación *.jar* y las librerías que nos hemos descargado mediante los archivos XML de Maven.

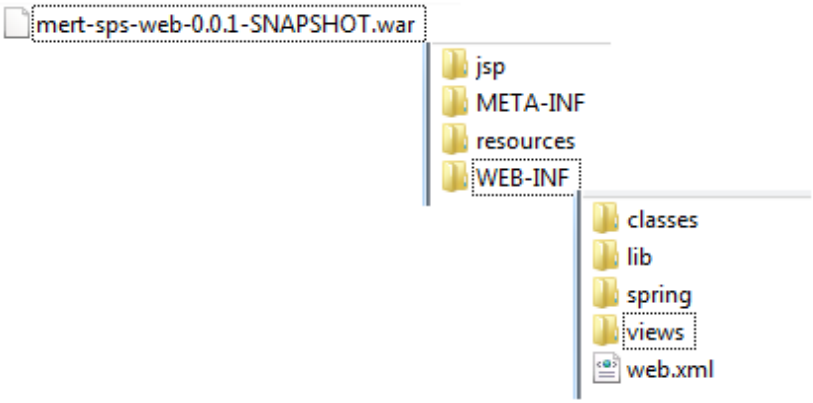


Ilustración 3.28: Paquete WAR del aplicativo

Este paquete archivo se ha de desplegar en el *Tomcat*. Una buena manera es a partir del aplicativo *Probe* que de una manera visual te permite hacer el despliegue de la aplicación. En la ilustración 3.29 se ve la interfaz gráfica. A partir de este punto solo hace falta seleccionar el archivo, darle un nombre si así se desea y presionar el botón de *deploy*.

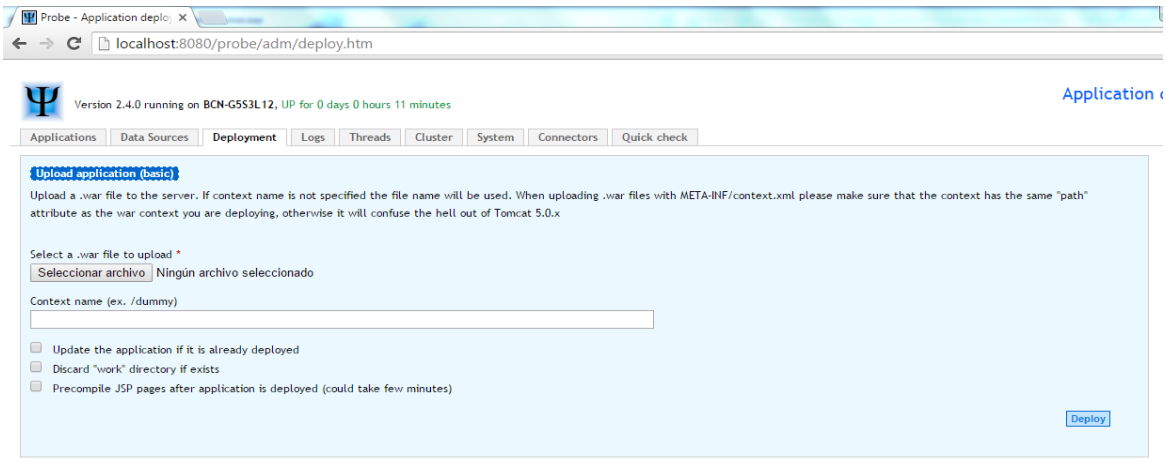


Ilustración 3.29: Aplicativo *Probe* desplegado en el *Tomcat*

Por otra parte, se ha de incluir la librería JRE de Java que se usa en el proyecto en el directorio principal del *Tomcat*. Por lo tanto, se incluye un archivo 'global.properties' que contiene variables que se usan en la aplicación en la carpeta 'conf' y se crea, si no existen, los archivos 'setenv.bat' (Windows) y 'setenv.sh' (Linux).

La ilustración 3.30 muestra el script para Linux que hay que introducir para que la aplicación lo encuentre.

```
1  export JAVA_HOME=$CATALINA_HOME/jre
2  export JAVA_OPTS="$JAVA_OPTS -Xms128m -Xmx512m -XX:MaxPermSize=256m -server -Dmert.sps=$CATALINA_HOME/conf"
```

Ilustración 3.30: Script para setear variables de entorno en Linux

Una vez hechas todas las modificaciones solo hace falta ejecutar el 'startup.bat' o 'startup.sh' para arrancar el servidor y acceder a la dirección del apache Tomcat seguido de /mert-sps-web/ para acceder a la aplicación desarrollada.

4. Pruebas de calidad

Una vez finalizada la aplicación se han llevado a cabo las pruebas pertinentes a la aplicación. De esta manera, primero se ha comprobado mediante la ejecución de los test unitarios que no hubiera fallos de concepto. Luego, se ha comprobado el rendimiento de MERT.

4.1. Test unitarios

Los test unitarios están programados en Java y tiene como objetivo comprobar que los métodos de las diferentes clases de la aplicación respondan tal y como se espera. Así que tenemos un test por cada método.

No se comprueba el completo funcionamiento de MERT, se prueban por separado cada una de las herramientas internas. Esto es útil ya que es fácil encontrar si el aplicativo falla en algún punto del código y por qué motivo lo hace.

El framework que se utiliza es JUnit. Que es un conjunto de clases que permiten realizar la ejecución de clases Java de manera controlada.

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"classpath:/spring-config/spring-persistence.xml"})
public class EnvironmentDataDatoTest {

    @Autowired
    EnvironmentDataDaoI environmentDataDaoI;

    @Test
    public void testFind(){

        EnvironmentData environmentData = new EnvironmentData();
        int i = environmentDataDaoI.findAll(EnvironmentData.class).size();
        if(i>0){environmentData = environmentDataDaoI.findAll(EnvironmentData.class).get(0);}
        Assert.assertTrue(i>0);
        Assert.assertNotNull(environmentData);
    }
    @Test
    public void testUpdate(){

        EnvironmentData environmentData = new EnvironmentData();
        EnvironmentData environmentData2= new EnvironmentData();
        //environmentDataDaoI.create(environmentData);
        int i = environmentDataDaoI.findAll(EnvironmentData.class).size();
        if(i>0){environmentData = environmentDataDaoI.findAll(EnvironmentData.class).get(0);
        String s = environmentData.getEquipo();
        environmentData.setEquipo("EQUIPO");
        environmentDataDaoI.update(environmentData);
        environmentData2 = environmentDataDaoI.findAll(EnvironmentData.class).get(0);
        }

        Assert.assertTrue(environmentData2.getEquipo().equals("EQUIPO"));
        Assert.assertTrue(i>0);
        Assert.assertNotNull(environmentData);
    }

    @Test
```

Ilustración 4.1: Test de la parte de persistencia

La ilustración 4.1 es un ejemplo de test unitario. En este test se determina primero con que clase se quiere realizar la prueba. En caso contrario, si no se selecciona una clase concreta, se elige la clase por defecto que es la 'BlockJUnit4ClassRunner'. Seguidamente se le pasa la ubicación del contexto de *Spring*. Por último, se utiliza *@Autowired* para extraer del contexto de Spring el *Bean*, en este caso del DAO de persistencia y *@Test* para marcar que métodos son de prueba.

Por último, se ejecuta el código mediante el framework de JUnit. El resultado es el mostrado en la ilustración 4.2, de manera que: si hay errores en el código los marca de color rojo, si no se cumple un test de prueba lo marca de color azul y si funciona correctamente se indica en color verde.

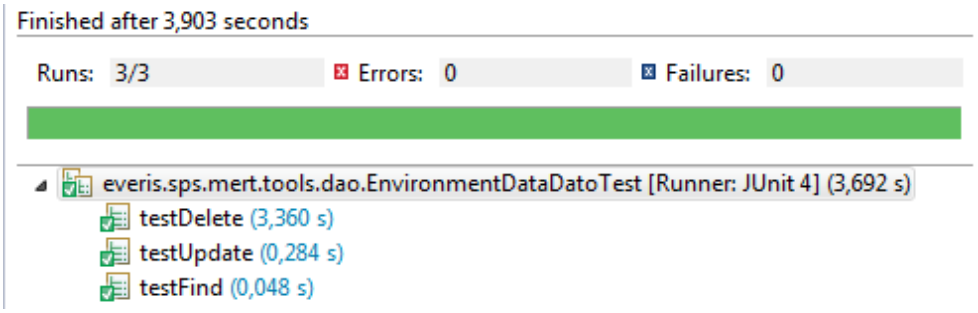


Ilustración 4.2: Ejecución test

4.2. Pruebas de rendimiento

Una vez se ha comprobado que la aplicación funciona correctamente, se ha de medir que mejora supone su utilización respecto al método anterior. Aunque de la manera que se ha abordado ya sugiere ciertas mejoras, hay que cerciorarse mediantes pruebas que muestren el tiempo que tarda en mostrar los resultados.

El estudio que se va a llevar a cabo se centra en las tres pantallas. De esta manera se lanza una petición HTTP para cada una de ellas, porque son los puntos donde la aplicación va a ir más lenta, debido a que se ejecutan más conexiones remotas.

En primer lugar, se ha de desplegar la aplicación en el *Tomcat*. Luego, se debe seleccionar *JMeter* como herramienta para simular el tráfico de varios usuarios simultáneamente. JMeter es una herramienta de carga para llevar a cabo simulaciones sobre cualquier recurso de Software. Hoy en día, esta herramienta no solo ofrece pruebas de componentes habilitados en Internet (HTTP) sino que permite también a Bases de Datos, programas Perl,… El único problema a priori es que no se pueden probar las líneas de código de JavaScript. Sin embargo, este inconveniente no es el punto que hace más lenta la aplicación.

Según los resultados obtenidos la aplicación mejora el tiempo de consulta de las versiones. En el caso más favorable (1 usuario concurrente) tarda en recibir la respuesta 6 segundos. Para el caso más desfavorable (10 usuarios concurrentes) la aplicación se demora 15 segundos. Así pues en ambos casos se obtiene una mejora más que significativa, que va a ahorrar tiempo en la supervisión del proyecto Zeus. El rango de tanto por ciento de mejora varia del 90% al 96%.

Para consultar la información más detallada sobre las pruebas de rendimiento, consultar el punto 6.5 del anexo, donde se muestran los resultados de las pruebas de rendimiento utilizando de 1 a 10 usuarios concurrentes. Adicionalmente se hace una prueba con 20 usuarios para comprobar su funcionamiento en un caso extremo.

El tiempo crítico viene marcado por la página principal, con 1 usuario la carga se demora unos 6 segundos. Para una carga de 20 usuarios el tiempo de carga es de aproximadamente 26 segundos (tiempo medio). Esto mejora en un 85% el tiempo que se tarda en consultar toda la información de las versiones.

5. Conclusiones

En este proyecto se ha desarrollado una aplicación Web. MERT muestra la información de cada uno de los entornos de una manera ordenada y ágil.

A la vista de los resultados obtenidos con las pruebas realizadas, se llega a la conclusión de que la aplicación es de gran utilidad para los equipos que necesitan conocer que contiene cada entorno, en especial a los equipos de *testing*. Al igual, que también es de utilidad, para los equipos que han desplegado una aplicación y necesitan saber si han desplegado la misma versión en cada una de las máquinas de un mismo entorno. Los puntos por los que es recomendado la utilización de la aplicación son:

- La velocidad a la que se accede a los datos. El aplicativo que ofrecía un servicio similar tardaba de 2 a 3 horas en recuperar toda la información. Se ha encontrado un camino más rápido para realizar estas consultas (6 segundos).
- La información que se muestra en las pantallas es información veraz, se ha implementado un código que lee la información de los despliegues de manera que se obtiene la versión real de cada despliegue. También es una información actualizada, MERT muestra el estado actual de los entornos.
- Se ha implementado para que el usuario final le resulte fácil entender la información que se le presenta, sin necesidad de que conozca como esta implementada o que procesos se producen en la aplicación. Es decir, cualquier usuario podría consultar la información que muestra el aplicativo.
- Permite conocer si se están siguiendo buenas prácticas dentro del proyecto Zeus. Se tiene pleno conocimiento de la ubicación de cada una de las versiones de Zeus.

Por último, como líneas futuras para aportar mayor funcionalidad al aplicativo se proponen las siguientes ideas:

- Delegar el mantenimiento del archivo ‘enviroment.txt’ a MERT. De esta manera con un sistema visual como el que se ha visto en la pantalla de edición se podría eliminar o añadir un entorno nuevo. Esto agilizaría la creación del archivo, y evitaría errores al no tener que preocuparse de la estructura.
- Lanzar tareas de diferente funcionalidad, que permitan interaccionar con las diferentes versiones desplegadas.
- Añadir nuevas funcionalidades de despliegues de una versión en un entorno.

- Filtrar la información que se muestra según el equipo del usuario que ingresa en el aplicativo.

Aunque estas dos últimas funcionalidades requerirían de añadir una página de *login*, de manera que no todo el mundo tenga acceso a accionar ciertas tareas.

6. Anexo

6.1. SPS (Sistema de Promoción del Software)

El Sistema de Promoción del Software (SPS) es un conjunto de normativas de desarrollo de la empresa, centradas en el versionado y la promoción del software. Focalizada en todos aquellos aspectos que conciernan a una entrega satisfactoria de elementos nuevos o alterados, así como su correcta y eficiente implantación y despliegue en un entorno gobernado por GNF (GasNatural Fenosa).

La normativa está enfocada según los distintos escenarios de trabaja en los que se puede ver inmerso un ciclo de construcción y mantenimiento de un proyecto de software (en los estadios de desarrollo y promoción), así como los actores identificados que participan dentro del proceso.

En lo concerniente a la fase de desarrollo, regula el contenido y formato de los elementos entregados que modelan la base de datos, propiedades de configuración y software. En este ámbito, aplicarán igualmente las políticas y convenciones a utilizar para poder almacenar adecuadamente los elementos en un repositorio de versionado, estableciéndose la nomenclatura de etiquetado y versionado en general.

En lo referente a la fase de promoción, regula las políticas de aplicación de los artefactos en los diferentes entornos de despliegue, así como las restricciones, requerimientos y métricas de validación que se impondrán sobre ellos.

6.2. Spring

Spring es un *framework* de aplicación para aplicaciones escritas en lenguaje de programación Java. *Spring* nace de una combinación de desarrollo de aplicaciones J2EE (*Java 2 Enterprise Edition*), incluyendo EJB (*Enterprise JavaBEANS*), *Servlets* y JSP *(Java Server Pages*), y se logró generar un solo paquete con una estructura más sólida y mejor soporte.

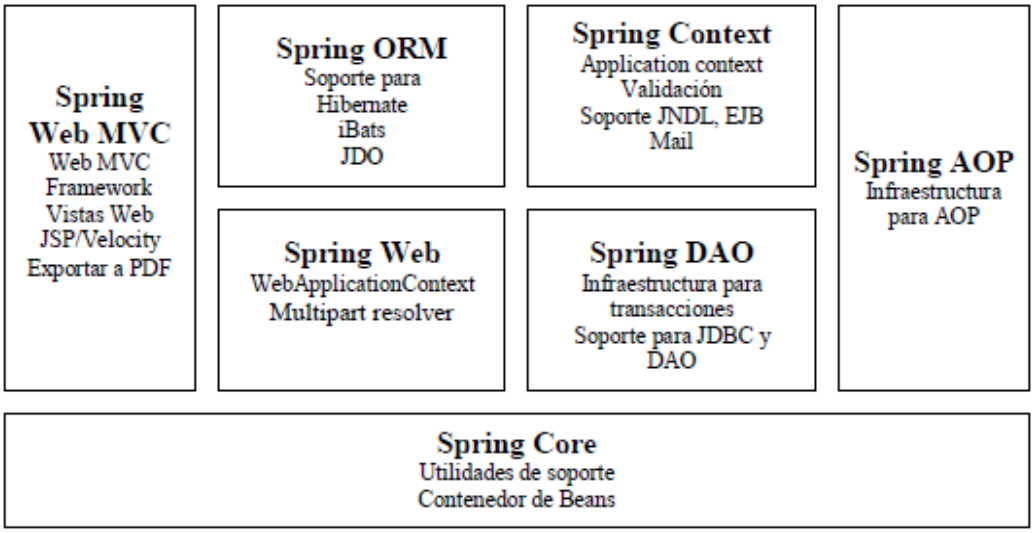


Ilustración 6.1: Arquitectura de Spring

Spring se compone de 7 módulos o capas, lo cual permite adaptarse a las necesidades de una aplicación.

- *Spring Core*: Es la parte que provee la funcionalidad esencial del *framework*, está compuesta por *BeanFactory*, el cual utiliza el patrón de Inversión de Control y configura los objetos a través de Inyección de Dependencias.
  - Los *Beans* son normalmente de tipo *Singleton*, que quiere decir que existe solo una instancia compartida.
- *Spring Context*: Es un archivo de configuración que provee de información contextual al *framework* general.
- *Spring AOP*: *Aspect-Oriented programming*, es una técnica que permite a los programadores modularizar ya se las preocupaciones *crosscutting* o el comportamiento que corta a través de las divisiones de responsabilidad, como *logging*, y manejo de transacciones.

- *Spring ORM*: *Object-Relational Mapping*, como alternativa a la implementación de JDBC se propone un módulo que soporta los *frameworks* ORM más populares como *Hibernate*.
- *Spring DAO*: *Data Access Object* es uno de los patrones más importantes y usados en aplicaciones J2EE, es la arquitectura de acceso a los datos.
- *Spring Web*: Se encuentra en la parte superior del módulo de contexto, y provee el contexto para las aplicaciones Web.
- *Spring Web MVC*: *Model View Controller*, Spring brinda un MVC para web bastante flexible y altamente configurable, pero esa flexibilidad no le quita sencillez, ya que se pueden desarrollar aplicaciones sencillas sin tener que configurar muchas opciones. Ofrece soporte para JSP.

Spring MVC

Características de *Spring MVC*:

- *Spring* hace una clara división entre controladores, modelos de *JavaBeans* y vistas.
- El MVC de *Spring* está basado en interfaces y es bastante flexible.
- Provee interceptores al igual que controladores.
- *Spring* no obliga a utilizar JSP como única tecnología *View* también se puede utilizar otras.
- Los controladores son configurados de la misma manera que los demás objetos en *Spring*, a través de IoC.
- Los Web *tiers* son más sencillos de probar que en otros *frameworks*.
- El Web *tiers* se vuelve una pequeña capa delgada que se encuentra encima de la capa de *business objects*.

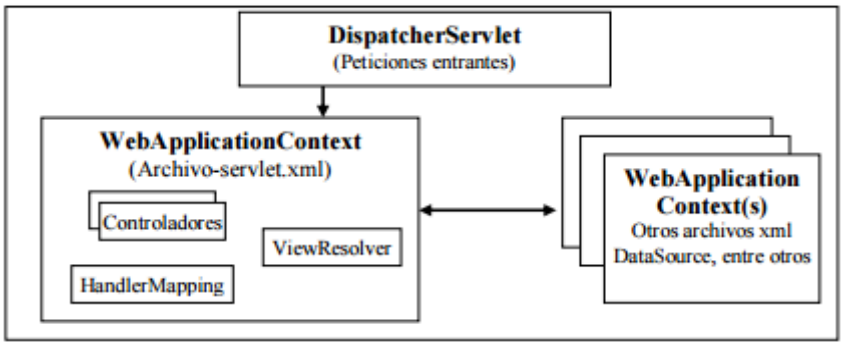


Ilustración 6.2: Arquitectura básica de *Spring MVC*

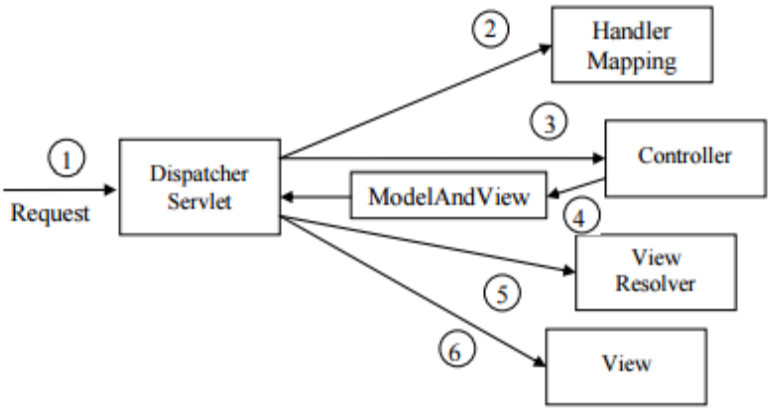


Ilustración 6.3: Ciclo de vida de un *request*

Para comprender cada parte de la arquitectura del *framework* de Spring MVC se presenta en la ilustración 6.3, el ciclo de vida de una petición o *request*.

1. El navegador manda una petición y lo recibe un *DispatcherServlet*.
2. Se debe escoger que *Controller* manejara la petición, para esto el *HandlerMapping* mapea los diferentes patrones de URL hacia los controladores, y se le regresa el *DispatcherServlet* el *Controller* elegido.
3. El *Controller* elegido toma la petición y ejecuta la tarea.
4. El *Controller* regresa un *ModelAndView* al *DispatcherServlet*.
5. Si el *ModelAndView* contiene un nombre lógico de un *View* se tiene que utilizar un *ViewResolver* para buscar ese objeto *View* que representa el *request* modificado.
6. Finalmente el *DispatcherServlet* despacha el *request* al *View*.

6.3. Prototipos pantallas

A continuación se presentan los prototipos que iban a diseñar antes del diseño definitivo. Al existir tanta información por cada entorno se optó por un modelo con muchas vistas cosa que complicaría al usuario final la visualización de la información.

Finalmente surgieron nuevos datos a mostrar y se convirtió en una solución inviable, por este motivo se ha optado por una solución más compacta que en apenas dos pantallas el usuario pueda visualizar los datos, más otra adicional para alterar algunos de los datos persistidos en BBDD.

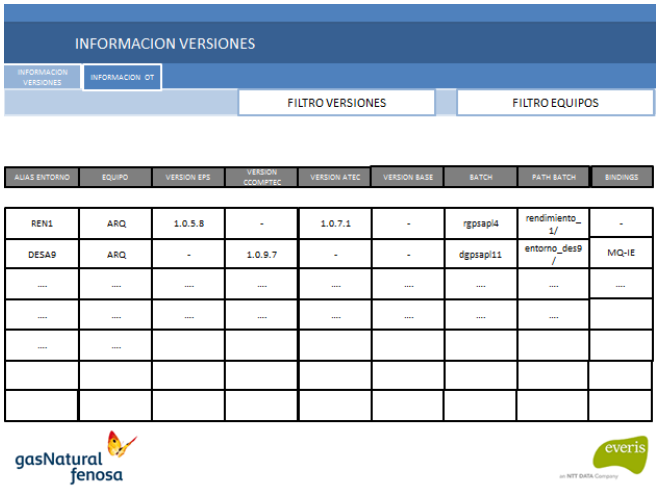


Ilustración 6.4: Información versiones por entorno

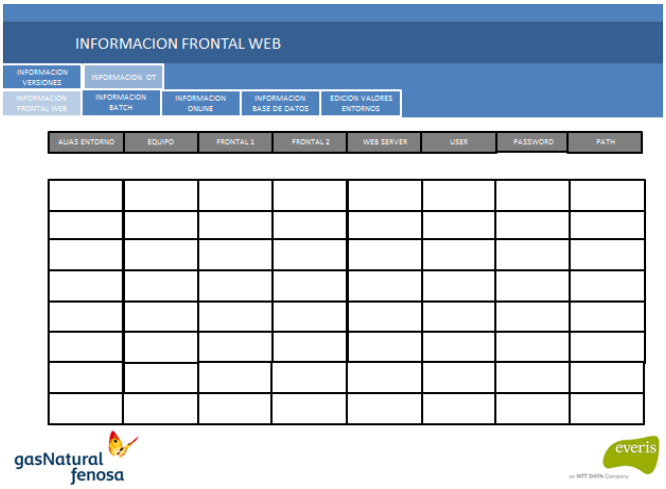


Ilustración 6.5: Información del frontal por cada entorno

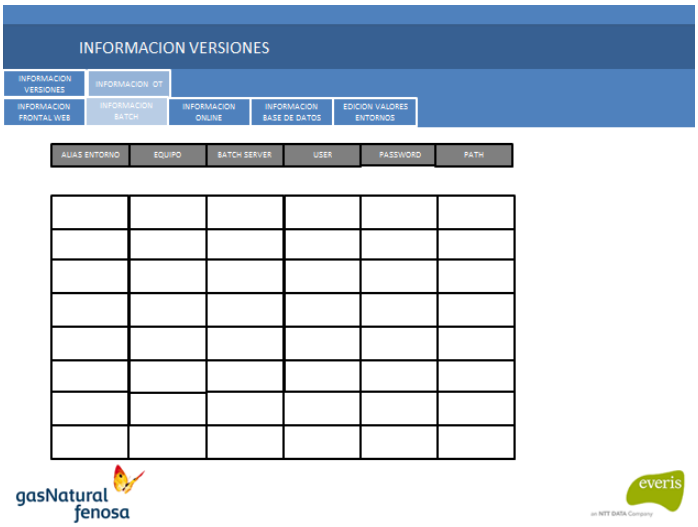


Ilustración 6.6: Información parte *Batch* de cada entorno

Ilustración 6.7: Información parte online de cada entorno

Ilustración 6.8: Información de las BBDD

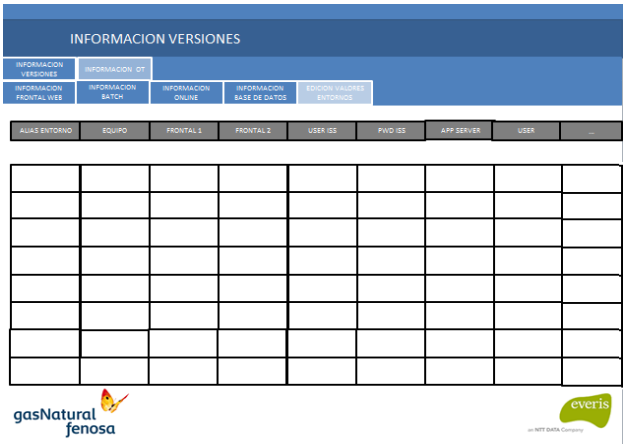


Ilustración 6.9: Modo de edición

6.4. Ciclo de Vida de un proyecto Maven

Existen 3 ciclos de vida en un proyecto *Maven*. *Clean*, donde se eliminan todas las clases compiladas y los archivos binarios generados del proyecto. *Default*, donde se generan los archivos binarios o artefactos. Por último tenemos *site*, que genera archivos HTML que describen nuestro proyecto.

Para ejecutarlos hemos de poner mvn 'ciclo' en la consola. En el caso del ciclo de vida del default hay 24 fases [26]. De las que cabe destacar las siguientes:

- *Validate*: Valida el proyecto.
- *Initialize*: Configura propiedades y crea directorios.
- *Compile*: Compila el código fuente del proyecto.
- *Test*: Ejecuta pruebas.
- *Package*: Genera el artefacto del proyecto.
- *Verify*: Verifica el artefacto generado.
- *Install*: Instala el artefacto en el repositorio local.
- *Deploy*: Sube el artefacto a un repositorio *Maven* en la red.

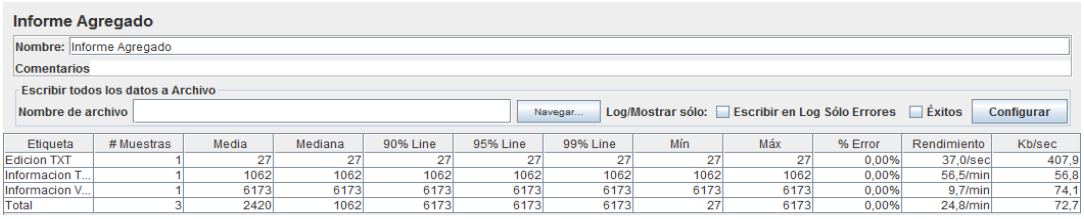
Existe la posibilidad de ejecutar únicamente una prueba, basta con escribir mvn 'fase'. También cabe tener en cuenta que cada fase contiene todas sus anteriores y que si falla una de las fases las demás no se completan, ya que se ejecutan en serie.

6.5. Pruebas de rendimiento

Estos son los datos obtenidos con JMeter para 1, 3, 6 y 10 usuarios concurrentes de las 3 pantallas del aplicativo, sin tener en cuenta las funcionalidades implementadas con JavaScript. Se añade también el análisis para 20 usuarios concurrentes.

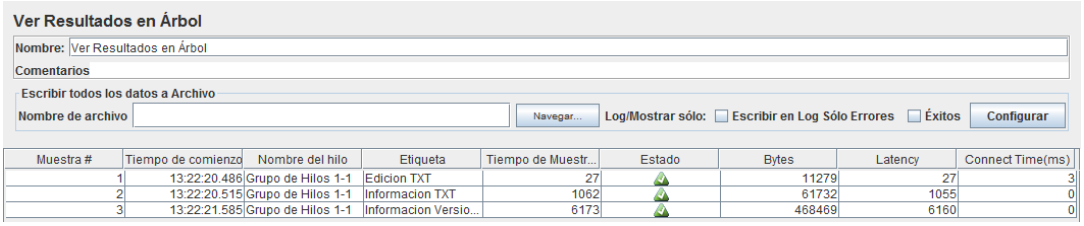
1 usuario concurrente

Como se observa en las informes de las consultas HTTP la pantalla de edición se muestra casi instantáneamente, la de información de las instancias se demora 1 segundo, en último lugar la pantalla principal (información versiones) tiene un tiempo de carga de 6 segundos.



Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec
Edicion TXT	1	27	27	27	27	27	27	27	0,00%	37,0/sec	407,9
Informacion T...	1	1062	1062	1062	1062	1062	1062	1062	0,00%	56,5/min	56,8
Informacion V...	1	6173	6173	6173	6173	6173	6173	6173	0,00%	9,7/min	74,1
Total	3	2420	1062	6173	6173	6173	27	6173	0,00%	24,8/min	72,7

Ilustración 6.10: Informe agregado 1 usuario



Muestra #	Tiempo de comienzo	Nombre del hilo	Etiqueta	Tiempo de Muestr...	Estado	Bytes	Latency	Connect Time(ms)
1	13:22:20.486	Grupo de Hilos 1-1	Edicion TXT	27		11279	27	3
2	13:22:20.515	Grupo de Hilos 1-1	Informacion TXT	1062		61732	1055	0
3	13:22:21.585	Grupo de Hilos 1-1	Informacion Versio...	6173		468469	6160	0

Ilustración 6.11: Informe resultados en árbol 1 usuario

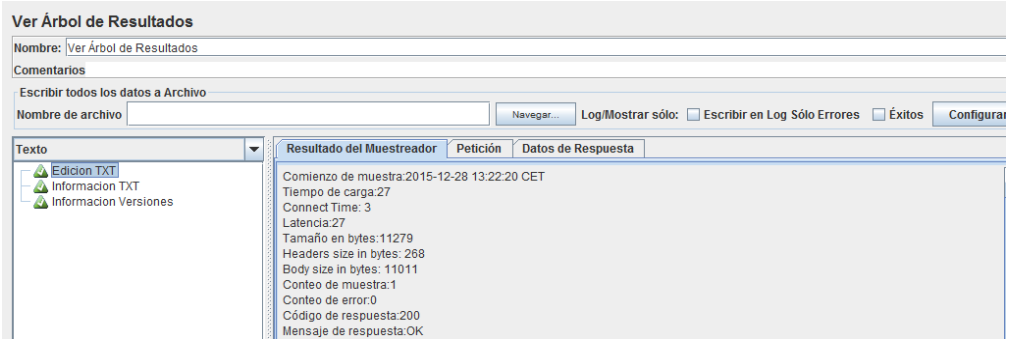


Ilustración 6.12: Informe árbol de resultados 1 usuario 'Edición TXT'

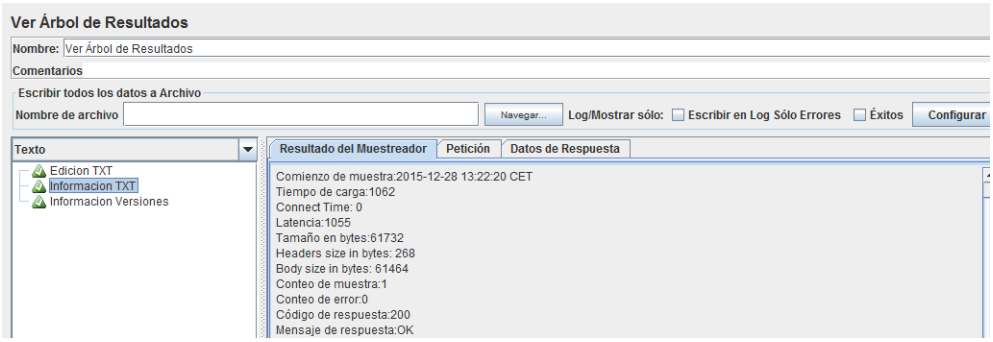


Ilustración 6.13: Informe árbol de resultados 1 usuario 'Información TXT'

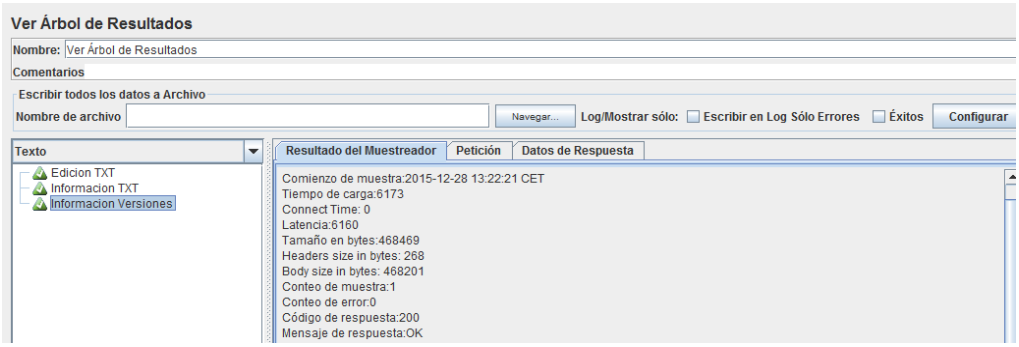
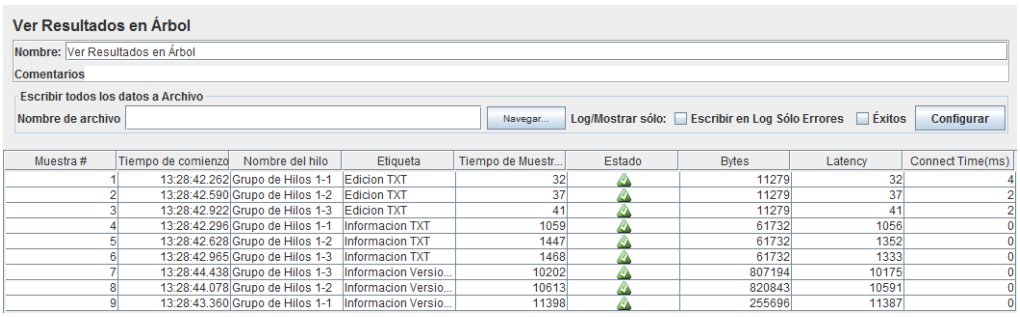


Ilustración 6.14: Informe árbol de resultados 1 usuario 'Información Versiones'

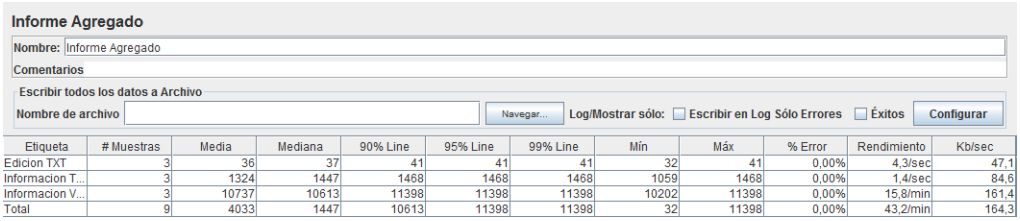
3 usuarios concurrentes

En la situación de 3 usuarios concurrentes el tiempo de carga de la aplicación aumenta. En el caso más relevante es en la pantalla de la información de las versiones ya que pasa de 6 segundos a 10 segundos.



Muestra #	Tiempo de comienzo	Nombre del hilo	Etiqueta	Tiempo de Muestr...	Estado	Bytes	Latency	Connect Time(ms)
1	13:28:42.262	Grupo de Hilos 1-1	Edicion TXT	32		11279	32	4
2	13:28:42.590	Grupo de Hilos 1-2	Edicion TXT	37		11279	37	2
3	13:28:42.922	Grupo de Hilos 1-3	Edicion TXT	41		11279	41	2
4	13:28:42.296	Grupo de Hilos 1-1	Informacion TXT	1059		61732	1056	0
5	13:28:42.628	Grupo de Hilos 1-2	Informacion TXT	1447		61732	1352	0
6	13:28:42.965	Grupo de Hilos 1-3	Informacion TXT	1468		61732	1333	0
7	13:28:44.438	Grupo de Hilos 1-3	Informacion Versio...	10202		807194	10175	0
8	13:28:44.078	Grupo de Hilos 1-2	Informacion Versio...	10613		820843	10591	0
9	13:28:43.360	Grupo de Hilos 1-1	Informacion Versio...	11398		255696	11387	0

Ilustración 6.15: Informe resultados en árbol 3 usuarios



Informe Agregado

Nombre: Informe Agregado

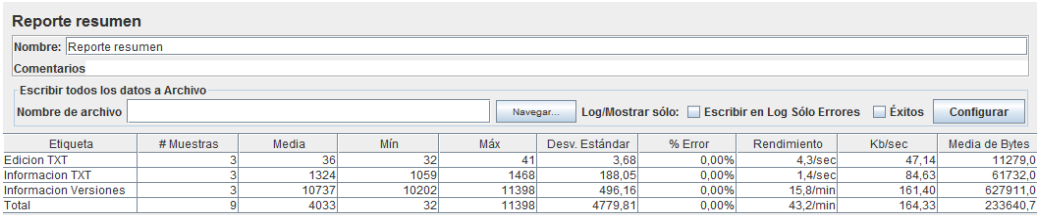
Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec
Edicion TXT	3	36	37	41	41	41	32	41	0,00%	4,3/sec	47,1
Informacion T...	3	1324	1447	1468	1468	1468	1059	1468	0,00%	1,4/sec	84,6
Informacion V...	3	10737	10613	11398	11398	11398	10202	11398	0,00%	15,8/min	161,4
Total	9	4033	1447	10613	11398	11398	32	11398	0,00%	43,2/min	164,3

Ilustración 6.16: Informe agregado 3 usuario



Reporte resumen

Nombre: Reporte resumen

Comentarios

Escribir todos los datos a Archivo

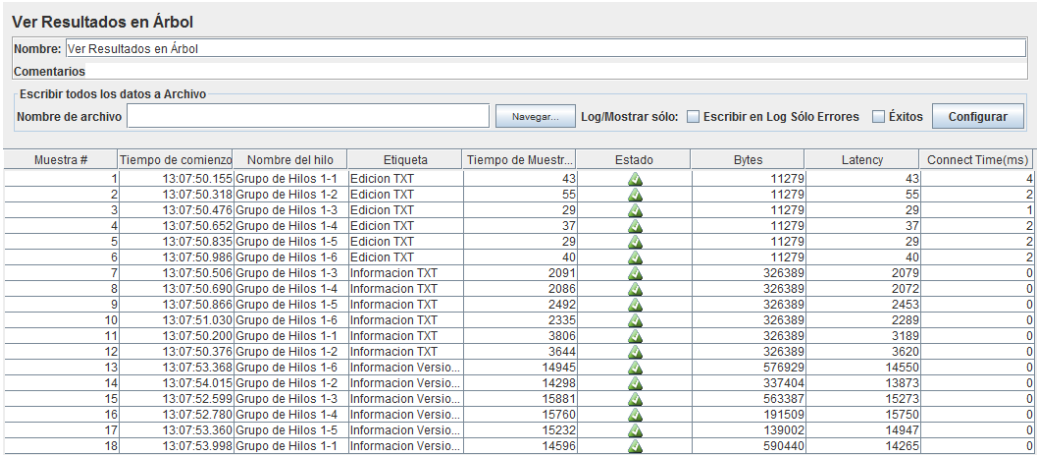
Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Edicion TXT	3	36	32	41	3,68	0,00%	4,3/sec	47,14	11279,0
Informacion TXT	3	1324	1059	1468	188,05	0,00%	1,4/sec	84,63	61732,0
Informacion Versiones	3	10737	10202	11398	496,16	0,00%	15,8/min	161,40	627911,0
Total	9	4033	32	11398	4779,81	0,00%	43,2/min	164,33	233640,7

Ilustración 6.17: Informe reporte resumen 3 usuarios

6 usuarios concurrentes

En el caso de 6 usuarios concurrentes vuelve a aumentar, aunque no se produce un incremento lineal, al doblar el número de usuarios respecto el caso anterior apenas se ha incrementado un 50% el tiempo de respuesta.



Ver Resultados en Árbol

Nombre: Ver Resultados en Árbol

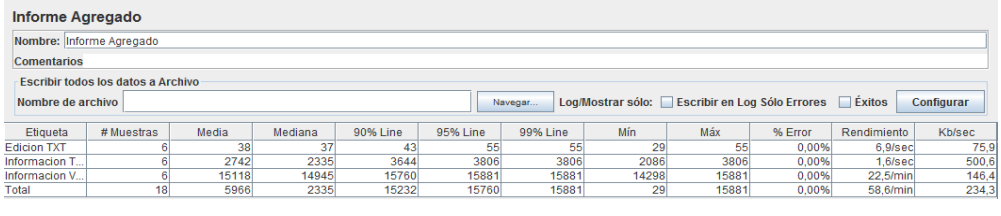
Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Muestra #	Tiempo de comienzo	Nombre del hilo	Etiqueta	Tiempo de Muestr...	Estado	Bytes	Latency	Connect Time(ms)
1	13:07:50.155	Grupo de Hilos 1-1	Edicion TXT	43		11279	43	4
2	13:07:50.318	Grupo de Hilos 1-2	Edicion TXT	55		11279	55	2
3	13:07:50.476	Grupo de Hilos 1-3	Edicion TXT	29		11279	29	1
4	13:07:50.652	Grupo de Hilos 1-4	Edicion TXT	37		11279	37	2
5	13:07:50.835	Grupo de Hilos 1-5	Edicion TXT	29		11279	29	2
6	13:07:50.986	Grupo de Hilos 1-6	Edicion TXT	40		11279	40	2
7	13:07:50.506	Grupo de Hilos 1-3	Informacion TXT	2091		326389	2079	0
8	13:07:50.690	Grupo de Hilos 1-4	Informacion TXT	2086		326389	2072	0
9	13:07:50.866	Grupo de Hilos 1-5	Informacion TXT	2492		326389	2453	0
10	13:07:51.030	Grupo de Hilos 1-6	Informacion TXT	2335		326389	2289	0
11	13:07:50.200	Grupo de Hilos 1-1	Informacion TXT	3806		326389	3189	0
12	13:07:50.376	Grupo de Hilos 1-2	Informacion TXT	3644		326389	3620	0
13	13:07:53.368	Grupo de Hilos 1-6	Informacion Versio...	14945		576929	14550	0
14	13:07:54.015	Grupo de Hilos 1-2	Informacion Versio...	14298		337404	13873	0
15	13:07:52.599	Grupo de Hilos 1-3	Informacion Versio...	15881		563387	15273	0
16	13:07:52.780	Grupo de Hilos 1-4	Informacion Versio...	15760		191509	15750	0
17	13:07:53.360	Grupo de Hilos 1-5	Informacion Versio...	15232		139002	14947	0
18	13:07:53.998	Grupo de Hilos 1-1	Informacion Versio...	14596		590440	14265	0

Ilustración 6.18: Informe resultados en árbol 6 usuarios



Informe Agregado

Nombre: Informe Agregado

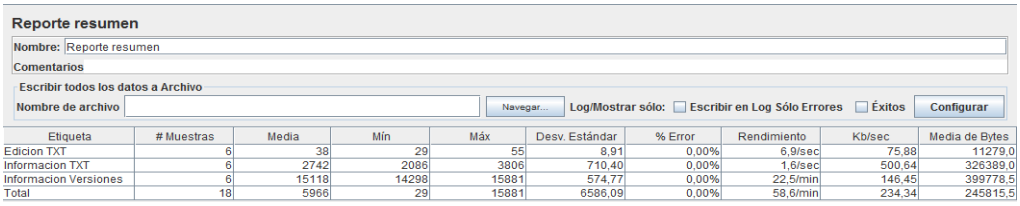
Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec
Edicion TXT	6	38	37	43	55	55	29	55	0,00%	6,9/sec	75,9
Informacion T...	6	2742	2335	3644	3806	3806	2086	3806	0,00%	1,6/sec	500,6
Informacion V...	6	15118	14945	15760	15881	15881	14298	15881	0,00%	22,5/min	146,4
Total	18	5966	2335	15232	15760	15881	29	15881	0,00%	58,6/min	234,3

Ilustración 6.19: Informe agregado 6 usuarios



Reporte resumen

Nombre: Reporte resumen

Comentarios

Escribir todos los datos a Archivo

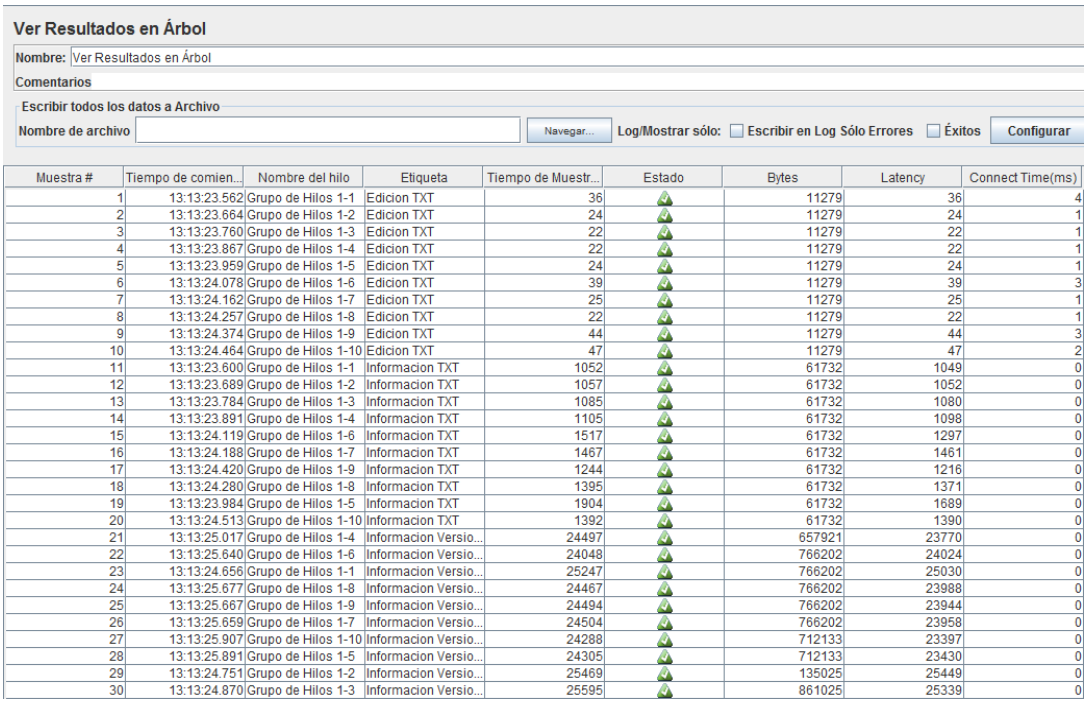
Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Edicion TXT	6	38	29	55	8,91	0,00%	6,9/sec	75,88	11279,0
Informacion TXT	6	2742	2086	3806	710,40	0,00%	1,6/sec	500,64	326389,0
Informacion Versiones	6	15118	14298	15881	574,77	0,00%	22,5/min	146,45	399778,5
Total	18	5966	29	15881	6586,09	0,00%	58,6/min	234,34	245815,5

Ilustración 6.20: Informe reporte resumen 6 usuarios

10 usuarios concurrentes

Para 10 usuarios tenemos que el tiempo de respuesta para la pantalla que supone un mayor coste computacional pasa de 15 segundos a aproximadamente 24 segundos.



Ver Resultados en Árbol

Nombre: Ver Resultados en Árbol

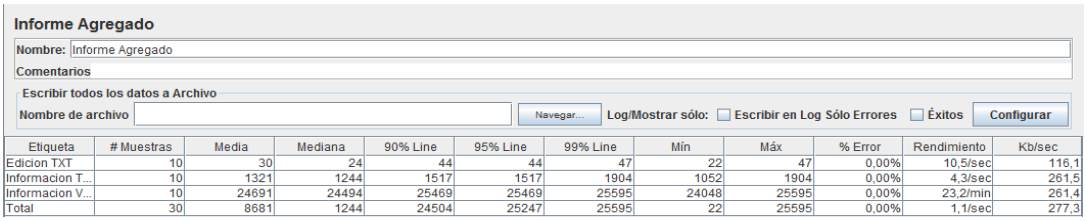
Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Muestra #	Tiempo de comien...	Nombre del hilo	Etiqueta	Tiempo de Muestr...	Estado	Bytes	Latency	Connect Time(ms)
1	13:13:23.562	Grupo de Hilos 1-1	Edicion TXT	36		11279	36	4
2	13:13:23.664	Grupo de Hilos 1-2	Edicion TXT	24		11279	24	1
3	13:13:23.760	Grupo de Hilos 1-3	Edicion TXT	22		11279	22	1
4	13:13:23.867	Grupo de Hilos 1-4	Edicion TXT	22		11279	22	1
5	13:13:23.959	Grupo de Hilos 1-5	Edicion TXT	24		11279	24	1
6	13:13:24.078	Grupo de Hilos 1-6	Edicion TXT	39		11279	39	3
7	13:13:24.162	Grupo de Hilos 1-7	Edicion TXT	25		11279	25	1
8	13:13:24.257	Grupo de Hilos 1-8	Edicion TXT	22		11279	22	1
9	13:13:24.374	Grupo de Hilos 1-9	Edicion TXT	44		11279	44	3
10	13:13:24.464	Grupo de Hilos 1-10	Edicion TXT	47		11279	47	2
11	13:13:23.600	Grupo de Hilos 1-1	Informacion TXT	1052		61732	1049	0
12	13:13:23.689	Grupo de Hilos 1-2	Informacion TXT	1057		61732	1052	0
13	13:13:23.784	Grupo de Hilos 1-3	Informacion TXT	1085		61732	1080	0
14	13:13:23.891	Grupo de Hilos 1-4	Informacion TXT	1105		61732	1098	0
15	13:13:24.119	Grupo de Hilos 1-6	Informacion TXT	1517		61732	1297	0
16	13:13:24.188	Grupo de Hilos 1-7	Informacion TXT	1467		61732	1461	0
17	13:13:24.420	Grupo de Hilos 1-9	Informacion TXT	1244		61732	1216	0
18	13:13:24.280	Grupo de Hilos 1-8	Informacion TXT	1395		61732	1371	0
19	13:13:23.984	Grupo de Hilos 1-5	Informacion TXT	1904		61732	1689	0
20	13:13:24.513	Grupo de Hilos 1-10	Informacion TXT	1392		61732	1390	0
21	13:13:25.017	Grupo de Hilos 1-4	Informacion Versio...	24497		657921	23770	0
22	13:13:25.640	Grupo de Hilos 1-6	Informacion Versio...	24048		766202	24024	0
23	13:13:24.656	Grupo de Hilos 1-1	Informacion Versio...	25247		766202	25030	0
24	13:13:25.677	Grupo de Hilos 1-8	Informacion Versio...	24467		766202	23988	0
25	13:13:25.667	Grupo de Hilos 1-9	Informacion Versio...	24494		766202	23944	0
26	13:13:25.659	Grupo de Hilos 1-7	Informacion Versio...	24504		766202	23958	0
27	13:13:25.907	Grupo de Hilos 1-10	Informacion Versio...	24288		712133	23397	0
28	13:13:25.891	Grupo de Hilos 1-5	Informacion Versio...	24305		712133	23430	0
29	13:13:24.751	Grupo de Hilos 1-2	Informacion Versio...	25469		135025	25449	0
30	13:13:24.870	Grupo de Hilos 1-3	Informacion Versio...	25595		861025	25339	0

Ilustración 6.21: Informe resultados en árbol 10 usuarios



Informe Agregado

Nombre: Informe Agregado

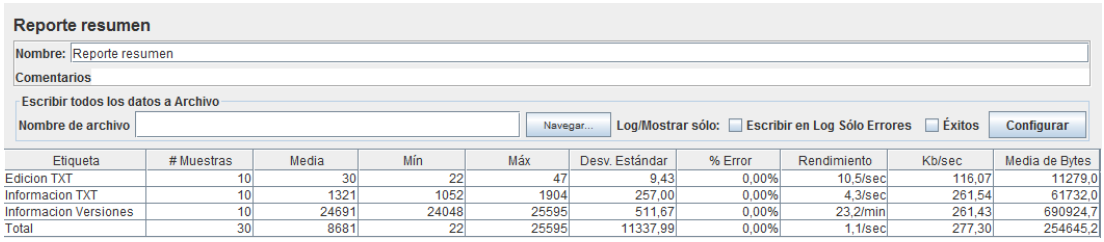
Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec
Edicion TXT	10	30	24	44	44	47	22	47	0,00%	10,5/sec	116,1
Informacion T...	10	1321	1244	1517	1517	1904	1052	1904	0,00%	4,3/sec	261,5
Informacion V...	10	24691	24494	25469	25469	25595	24048	25595	0,00%	23,2/min	261,4
Total	30	8681	1244	24504	25247	25595	22	25595	0,00%	1,1/sec	277,3

Ilustración 6.22: Informe agregado 10 usuario



Reporte resumen

Nombre: Reporte resumen

Comentarios

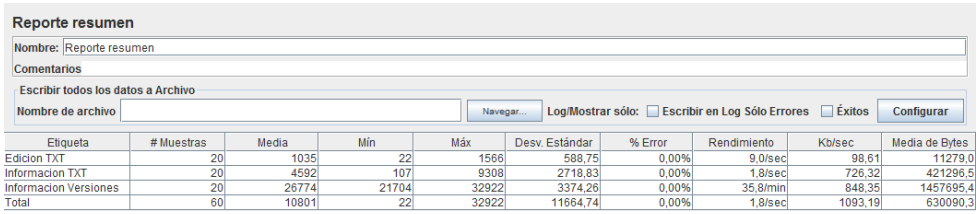
Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Edicion TXT	10	30	22	47	9,43	0,00%	10,5/sec	116,07	11279,0
Informacion TXT	10	1321	1052	1904	257,00	0,00%	4,3/sec	261,54	61732,0
Informacion Versiones	10	24691	24048	25595	511,67	0,00%	23,2/min	261,43	690924,7
Total	30	8681	22	25595	11337,99	0,00%	1,1/sec	277,30	254645,2

Ilustración 6.23: Informe reporte resumen 10 usuarios

20 usuarios concurrentes



Reporte resumen

Nombre: Reporte resumen

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Log/Mostrar sólo: Escribir en Log Sólo Errores | Éxitos | Configurar

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Edicion TXT	20	1035	22	1566	588,75	0,00%	9,0/sec	98,61	11279,0
Informacion TXT	20	4592	107	9308	2718,83	0,00%	1,8/sec	726,32	421296,5
Informacion Versiones	20	26774	21704	32922	3374,26	0,00%	35,8/min	848,35	1457695,4
Total	60	10801	22	32922	11664,74	0,00%	1,8/sec	1093,19	630090,3

Ilustración 6.24: Informe reporte resumen 20 usuarios

La aplicación MERT no está pensada para grandes cargas de usuarios debido a los procesos que se ejecutan en ella. En total unos 15 usuarios van a poder lanzar peticiones a la aplicación y aunque el aplicativo puede dar respuesta de una manera satisfactoria a más de 20 usuarios concurrentes, no se va a dar tal caso. Lo normal es que al mismo tiempo estén entre 1 y 3 usuarios. Podemos decir que en este caso el tiempo de demora del aplicativo ronda los 7-8 segundos.

7. Referencias

[1] Eclipse, http://www.eclipse.org/ (revisado 19/12/2015)

[2] SVN, https://subversion.apache.org/ (revisado 19/12/2015)

[3] Apache Tomcat, http://tomcat.apache.org/ (revisado 19/12/2015)

[4] Hudson, https://es.wikipedia.org/wiki/Hudson_(software) (revisado 19/12/2015)

[5] Apache Ant, http://ant.apache.org/ (revisado 19/12/2015)

[6] Apache Maven, https://maven.apache.org/ (revisado 19/12/2015)

[7] Weblogic, http://oracle.com/technetwork/middleware/weblogic/overview/index-085209.html (revisado 19/12/2015)

[8] Java Development Kit (JDK), http://oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html (revisado 19/12/2015)

[9] SSH, http://web.mit.edu/rhel-doc/3/rhel-rg-es-3/ch-ssh.html (revisado 19/12/2015)

[10] File Transfer Protocol (FTP), https://es.wikipedia.org/wiki/File_Transfer_Protocol/ (revisado 19/12/2015)

[11] Java Management Extensions (JMX), https://docs.oracle.com/cd/E13222_01/wls/docs90/jmx/accessWLS.html/ (revisado 19/12/2015)

[12] RMI-T3 JMX, http://docs.oracle.com/cd/E14571_01/web.1111/e13721/rmi_t3.htm#WLRMI143/ (revisado 19/12/2015)

[13] SOAP, http://www.w3.org/TR/soap/ (revisado 20/12/2015)

[14] REST, http://www.w3.org/Submission/SA-REST/ (revisado 20/12/2015)

[15] Spring MVC, http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html (revisado 19/12/2015), http://zeroturnaround.com/rebellabs/top-4-java-web-frameworks-revealed-real-life-usage-data-of-spring-mvc-vaadin-gwt-and-jsf/ (revisado 19/12/2015)

[16] Java Server Faces (JSF), https://es.wikipedia.org/wiki/JavaServer_Faces (revisado 19/12/2015)

[17] Vaadin, https://vaadin.com/book/es/-/page/preface.html (revisado 19/12/2015)

[18] Hibernate, https://docs.jboss.org/hibernate/orm/3.5/reference/en/html/ (revisado 19/12/2015)

[19] Maven (MVN), http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/322 (revisado 19/12/2015)

[20] Junit, https://elrincondeaj.wordpress.com/2012/07/09/junit/ (revisado 19/12/2015)

[21] Java, http://www3.uji.es/~belfern/pdidoc/IX26/Documentos/introJava.pdf (revisado 19/12/2015)

[22] Hyper Text Markup Language (HTML5), http://www.w3schools.com/html/html5_intro.asp (revisado 19/12/2015)

[23] Cascading Style Sheets (CSS), http://www.w3schools.com/css/ (revisado 19/12/2015)

[24]JavaScript (JS), http://www.w3schools.com/js/ (revisado 19/12/2015) (revisado 19/12/2015)

[25] Fases de un mvn default, http://maven.apache.org/guides/introduction/introduction-tothe-lifecycle.html (revisado 19/12/2015)