Creación de portales para B2C:

David Palomar Delgado [1]

[1] University Carlos III – Calle Madrid, 126, 28903 Getafe, Madrid, Spain
dpalomar@inf.uc3m.es

Resumen: En este capítulo ofrecemos una panorámica general de dos productos comerciales muy utilizados actualmente en el desarrollo profesional de portales y sitios Web de comercio electrónico. Aunque estos productos actualmente también proporcionan soporte para desarrollar procesos de B2B, aquí nos centraremos fundamentalmente en el desarrollo de portales mediante com-ponentes preparados para su reutilización (a veces llamados portlets). La visión sucinta pro-porcionada en este documento puede complementarse con el uso de licencias de evaluación de los productos que se tratan, que cuentan con tutoriales que dan una idea más práctica de las funcionalidades proporcionadas. En el apartado de "Recursos en la Web" pueden encontrarse las direcciones Web de estos productos. Dentro del mundo Java existen otras alternativas opensource para la creación de portales que contienen todos los servicios anteriormente mencionados por las otras plataformas y que cumplen los estándares de industria, y que son alternativas perfectamente válidas para un pro-yecto de grandes prestaciones. En este capítulo se revisarán las alternativas existentes.

Palabras clave: B2B; portales web

Abstract. In this chapter we offer an overview of two commercial products currently widely used in the professional development of e-commerce portals and websites. Although these products currently also provide support for developing B2B processes, here we will focus primarily on the development of portals using reusable speakers (sometimes called portlets). The succinct vision provided in this document can be complemented with the use of evaluation licenses for the products in question, which include tutorials that give a more practical idea of the functionalities provided. The Web addresses of these products can be found in the "Web Resources" section. Within the Java world there are other opensource alternatives for the creation of portals that contain all the services previously mentioned by the other platforms and that meet industry standards, and that are perfectly valid alternatives for a project of great features. In this chapter we will review the existing alternatives.

Keywords: B2B; web sites

Introducción

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Los grandes sitios de comercio electrónico y los portales Web gestionan una enorme cantidad de páginas dinámicas, y dan soporte a procesos de negocio B2C que requieren varios pasos para su realización, y que deben estar adecuadamente soportados por procesos automatizados para ser eficientes. La complejidad y dificultad de mantenimiento de este tipo de aplicaciones requiere de software especializado, ya que "partir de cero" en estos casos supone un esfuerzo de desarrollo tan grande que difícilmente encaja en los calendarios de desarrollo. Por otro lado, las funcionalidades y servicios proporcionados por estas aplicaciones son **muy similares** de un sistema a otro, lo cual facilita la reutilización de componentes y el uso de herramientas uniformes. Por todo ello, es importante conocer las posibilidades que nos ofrecen los paquetes comerciales a la hora de decidir sobre su uso y de estimar el esfuerzo necesario para desarrollar sistemas de comercio electrónico utilizándolos.
En este tema ofrecemos una panorámica general de dos productos comerciales muy utilizados actualmente en el desarrollo profesional de portales y sitios Web de comercio electrónico. Aunque estos productos actualmente también proporcionan soporte para desarrollar procesos de B2B, aquí nos centraremos fundamentalmente en el desarrollo de portales mediante componentes preparados para su reutilización (a veces llamados *portlets*). La visión sucinta proporcionada en este documento puede complementarse con el uso de licencias de evaluación de los productos que se tratan, que cuentan con tutoriales que dan una idea más práctica de las funcionalidades proporcionadas. En el apartado de "Recursos en la Web" pueden encontrarse las direcciones Web de estos productos.
Ambas plataformas están basadas en el estándar J2EE de Java, y por tanto, descansan en el soporte de los *Enterprise Java Beans* (EJB) – véase (Roman, Ambler & Jewell, 2001) – aunque los servicios que proporcionan y su arquitectura varían de manera significativa, como se verá a lo largo de este tema [1-7].

La plataforma j2ee.

J2EE comprende un conjunto de APIs que amplían y mejoran el modelo J2SE (Java 2 Standar Edition), orientado su funcionalidad fundamentalmente hacia aplicaciones distribuidas. Pero además J2EE proporciona una infraestructura de entorno de ejecución para albergar y administrar aplicaciones, típicamente hablamos del servidor donde se ejecutan las aplicaciones, la especificación no dice como debe estar construida la infraestructura de ejecución, en su lugar define unos roles e interfaces que separan claramente la infraestructura y las aplicaciones que se ejecutan en ella. Define el concepto de contenedor y especifica lo que llama un contrato entre el contenedor y las aplicaciones. En la especificación entra a jugar parte de la gran mayoría de las empresas relevantes en las tecnologías de la información, la especificación ha pasado de ser propiedad de uno solo a ser un conjunto de normas y "Best Practices" proporcionadas por todas las industrias del sector, aportando su conocimiento y manera de trabajo. Esto ha permitido que distintas empresas jueguen con un conjunto de reglas comunes, ya que la especificación no dicta como debe construirse la infraestructura, los distintos fabricantes personalizan con su mejor conocimiento las herramientas que dan soporte a la especificación.

A continuación, se muestran las APIs que entran a formar parte de la especificación:

1. JDBC 3.0: API que permite de manera estándar la conexión con distintas fuentes de datos. Proporcionando un pool de conexiones.
2. EJB 3.0: Componentes distribuidos y robustos, de lógica de negocio.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

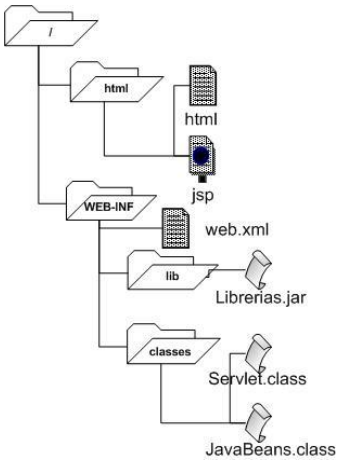
3. Java Servlets 2.4: Abstracción OO para construir aplicaciones Web.
4. JSP 2.0: Extensión de la anterior basada en plantillas.
5. JMS 1.0.1: Mensajes asíncronos. (Message Oriented Middleware).
6. JTA 1.0: Transacciones distribuidas.
7. JavaMail 1.2: Proporciona toda la funcionalidad para emisión y recepción de datos mediante correo electrónico. Soporte a IMAP, SMTP.
8. Java XML Pack: Conjunto de librerías para el tratamiento de la información en formato XML, este pack contiene:
    a. Java API for XML Processing (JAXP): procesa documentos usando varios parsers.
    b. Java Architecture for XML Binding (JAXB): Procesa documentos XML usando un Schema como resultado de componentes JavaBeans.
    c. Java API for XML-based RPC (JAX-RPC) – Permite enviar llamadas a métodos de componentes remotos usando SOAP sobre Internet y recibir los resultados.
    d. Java API for XML Messaging (JAXM) – Permite enviar mensajes SOAP sobre Internet de una manera estándar.
    e. Java API for XML Registries (JAXR) – Proporciona una manera estándar de acceder a registros de negocio para compartir información.
9. Java IDL: Proporciona la infraestructura para la integración con servicios CORBA.
10. RMI-IIOP: Permite la comunicación remota de objetos, y es un API base para otras tecnologías.
11. JNDI: Permite la conexión a servicios de nombres de una forma estándar.
12. JCA 1.0: Conectores para la integración con Sistemas de Información (EIS).

El contenedor.

La especificación J2EE propone un entorno bajo el cual se asegura un comportamiento de los componentes de aplicación igual para distintos entornos, el modelo del contenedor contiene recursos necesarios para establecer mecanismos de seguridad, transacciones, gestión de memoria, concurrencia, sincronización de tareas y conexiones con otros sistemas de manera homogénea para todas los aplicaciones, independientemente de la plataforma donde sean ejecutadas [8-11].
El lenguaje Java, proporciona los medios necesarios para que una misma aplicación pueda ejecutarse en distintas plataformas, pero una aplicación tiene una serie de requisitos que no inciden directamente en la funcionalidad pero que deben ser contemplados, como por ejemplo la seguridad, la gestión de memoria, y la configuración del acceso a los datos. Además una aplicación contiene una serie de recursos que pueden ser exclusivos de la aplicación en si, como conexiones con base de datos, servi-

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

cios de nombres, archivos de configuración, etcétera. Resulta difícil ejecutar una aplicación en distintas plataformas si tiene integrado en su código la gestión de todos estos recursos, sin tener que adaptar esa aplicación a la nueva plataforma y sin recompilar la aplicación de nuevo.
Por ello, la especificación J2EE proporciona el modelo del contenedor, de manera que cualquier aplicación que siga las normas que dicta la especificación pueda ser desplegada en cualquier contenedor que cumpla también la especificación, independientemente de la plataforma. La especificación expone las normas que deben cumplir los proyectos J2EE y el contenedor, esto es necesario porque el contenedor debe tener acceso a todos los recursos del proyecto para poder gestionarlos. Este modelo permite que el desarrollo se centre en el aspecto funcional, dejando las tareas de bajo nivel gestionadas por el contenedor, se produce un modelo de programación declarativa. La unión entre el contenedor y la aplicación se realiza mediante el llamado contrato, que es un fichero de configuración dependiente del proyecto, denominado Deployment Descriptor y cuyo formato es XML [12-15].
Aún queda un asunto por resolver, ya que para que los proyectos puedan ser distribuidos entre distintos contenedores deben crearse manteniendo una estructura común de directorios y directivas de configuración, para que de esta manera independientemente de la forma de trabajo del equipo de desarrollo, cualquier contenedor pueda acceder y gestionar los recursos. La estructura de un proyecto web en la especificación J2EE se expone a continuación:



53.Figura 15. Estructura de proyecto web.

Como se aprecia en la figura todo proyecto web contiene un directorio raíz del cual dependen una serie de directorios, inicialmente el contenido estático como páginas html, imágenes o multimedia dependen del directorio raíz o en su defecto de uno creado a tal fin, los documentos jsp también se incorporan aquí. A continuación y de manera obligatoria debe existir un directorio llamado WEB-INF donde se gestionarán todos los recursos dinámicos del proyecto, aquí se insertará también el descriptor de despliegue (Deployment Descriptor) que configura todos los componentes del proyecto, debajo contendrá obligatoriamente dos directorios llamados lib y classes el primero almacenará las librerías externas que usemos en el proyecto, y en el segundo se encontrarán los componentes desarrollados como servlets, javabeans y librerías de etiquetas. Con esta estructura homogénea la especificación se asegura de que un proyecto pueda ser desplegado en distintos contenedores y estos sepan tratar la misma información por igual.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Para poder distribuir de manera homogénea una aplicación, la especificación J2EE propone tres tipos de empaquetamiento de las clases y recursos que forman parte de la aplicación.

1. Ficheros JAR (Java Archive)
2. Ficheros WAR (Web Archive)
3. Ficheros EAR (Enterprise Archive)

El primero de ellos se usa para almacenar componentes que se reutilizarán en las aplicaciones, típicamente clases, ficheros de configuración y otros recursos, en una aplicación J2EE este tipo de ficheros se usa para la generación de Enterprise JavaBeans (EJB). Los archivos WAR, permiten empaquetar todos los recursos y estructura de directorios visto anteriormente, se usan para la creación y distribución de proyectos web, por último el tipo EAR permite empaquetar una aplicación mucho más robusta que haga uso de componentes web y EJBs juntos, de esta manera un EAR permite juntar en un único archivo ficheros WAR y ficheros JAR [16-23].

El descriptor de despliegue.

La especificación J2EE proporciona un modelo de programación declarativa, permitiendo al programador centrarse en desarrollar las funcionalidades de la aplicación y dejando tareas de bajo nivel como puede ser la gestión de la memoria, conexión con repositorios de datos, sincronización, concurrencia, transacciones, multi-threading, seguridad, en manos del contenedor.

La especificación nombra como contrato a la relación que existe entre los distintos componentes que forman parte de una aplicación J2EE y el contenedor que la soporta. Esta relación puede venir en base a alguna de las tareas de bajo nivel expuestas anteriormente, o bien, como la especificación dicta que un contenedor web debe dar soporte completo al protocolo http, el contenedor web puede por lo tanto funcionar como un servidor web aunque no sea este su cometido, por lo que esta relación puede venir en forma de alias, mapping de directorios virtuales, paso de parámetros de configuración, etc.

Este contrato viene definido mediante un documento en formato XML, denominado Deployment Descriptor o descriptor de despliegue, básicamente la tarea de configurar el proyecto mediante el descriptor será realizada o bien por el programador encargado del proyecto o por un perfil administrador, también denominado “deployer” que normalmente mediante una consola de administración del servidor de aplicaciones podrá configurar las directivas necesarias para el proyecto, aunque también es posible generar estas directivas de manera manual editando el propio descriptor. Ya que el descriptor es un documento XML debe cumplir con las normas de este lenguaje de marcado y además contiene un DTD que especifica la gramática que soporta. Todo descriptor de despliegue de un proyecto web comienza con un nodo raíz denominado <web-app>, bajo el cual se irán indicando las distintas etiquetas necesarias para configurar el proyecto, por ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
        <servlet>
                <servlet-name>Manager</servlet-name>
                <servlet-class>org.apache.catalina.servlets.ManagerServlet</servlet-class>
                <init-param>
                        <param-name>debug</param-name>
                        <param-value>2</param-value>
                </init-param>
        </servlet>
        <servlet-mapping>
                <servlet-name>Manager</servlet-name>
                <url-pattern>/*</url-pattern>
        </servlet-mapping>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        <security-constraint>
                <web-resource-collection>
                        <web-resource-name>Entire Application</web-resource-name>
                        <url-pattern>/*</url-pattern>
                </web-resource-collection>
                <auth-constraint>
                        <role-name>manager</role-name>
                </auth-constraint>
        </security-constraint>
        <login-config>
                <auth-method>BASIC</auth-method>
                <realm-name>Tomcat Manager Application</realm-name>
        </login-config>
</web-app>
```

En este ejemplo de un descriptor web.xml, se aprecia como en esta aplicación web exite un servlet especificado por su paquete de clases dentro de la etiqueta <servlet-class> al cual se establece un alias denominándolo Manager, además se establecen unos parámetros de inicio con las estiquetas <init-param> y <param-value>. Con la etiqueta <servlet-mapping> se establece una unión en las que todas las llamadas que se realicen a la URL raíz dentro de ese servidor (indicado por /*), se pasen automáticamente al servlet Manager. A continuación se configura la seguridad en el proyecto, estableciendo un tipo de autenticación básica con un reino de seguridad denominado Tomcat Manager Application y que sólo estará disponible para el rol de manager, esta seguridad se aplica a toda la aplicación nuevamente determinando a todas las peticiones que se realicen a la URL raíz [24-27].

La especificación servlet.

En las arquitecturas web tradicionales la necesidad de recibir y manipular información proveniente del usuario fomentó la aparición de el modelo CGI, el cual permitía usar las características de comunicación del protocolo http para recibir la información del usuario mediante peticiones y parámetros, realizar la ejecución de un programa con su lógica de negocio en el servidor, y después hacer las operaciones para provocar una respuesta enviada de vuelta al cliente. Inicialmente los lenguajes soportados para la creación de programas que realizaran estas tareas y se ejecutarán en el servidor fueron lenguajes como Perl, C o C++. La necesidad de manipular la información del usuario y disponer de contenido dinámico hizo que la plataforma Java pasara de ejecutarse en las plataformas de los clientes mediante los denominados applets a copiar el modelo CGI y ejecutarse en el servidor mediante nuevo modelo de componentes denominados Servlets.

Los Servlets copian el modelo CGI proporcionando una infraestructura para el trabajo mediante el protocolo http, proporcionando un API completo de desarrollo, y la especificación J2EE proporciona el modelo de contenedor, mediante el cual asegura la portabilidad de los proyectos web. Los Servlets proporcionan mejoras sobre el modelo de CGI clásico, pero también heredan sus carencias, como por ejemplo: la unión en un mismo programa de lógica de negocio, y lógica de presentación haciendo las aplicaciones menos escalables y mantenibles. En cuanto a las mejoras, proporcionan un sistema de seguridad diferente al de CGI clásico, el cual estaba basado en permisos de ejecución sobre directorios, algo muy susceptible a la incorporación de código malicioso, los Servlets no disponen su seguridad en base a permisos de ejecución, sino que se ejecutarán bajo una infraestructura de seguridad de la máquina virtual. Otra ventaja es que en el modelo CGI clásico se generaba un nuevo proceso por cada petición de un cliente, mientras que en el modelo Servlets se genera un único proceso y se sirven múltiples instancias, una por cada cliente [28-30].

Un servlet puede considerarse como una especie de mini-servidor especializado, escrito en Java, que se encarga de recibir la petición, generar contenido dinámicamente y, finalmente, devolver la respuesta. Los servlets se ejecutan dentro de un contenedor web (típicamente conocido como motor de servlets) que se encarga de traducir las peticiones nativas del protocolo a objetos Java y las respuestas en forma de objetos Java a respuestas nativas del protocolo.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

La especificación Servlet define un ciclo de vida de los componentes, así de esta manera se asegura de que todos los componentes se escriban y se ejecuten de la misma forma, independientemente del contenedor que les dé soporte. El contenedor se encarga de gestionar el ciclo de vida del servlet, así como de proporcionarle los servicios que necesite. El servidor redirige las peticiones al contenedor para que éste, a su vez, las delegue en el servlet correspondiente. En el caso del servidor web, este comportamiento se configura nativamente y, en el caso del contenedor, mediante un mecanismo estándar definido en la especificación Servlet.

Según la especificación Servlet el contenedor web debe implementar las funciones del servidor web, sin embargo, este efecto se consigue habitualmente conectando un servidor web con el contenedor, por ejemplo Apache + JBoss. Por tanto puede considerarse al contenedor web como un servidor web que debe llevar a cabo una serie de tareas tales como: proporcionar servicios de red para el establecimiento de peticiones y respuestas mediante http, codificar y decodificar peticiones y respuestas en formato MIME, configurar los servlets en función de los descriptores de despliegue y gestionar el ciclo de vida de los servlets. Además hay otras características de los contenedores que la especificación no exige pero que recomienda, en la mayoría de los casos estas características se pueden dar por presentes, como dar soporte al protocolo HTTP 1.1, soporte a HTTPS, y r estricciones de seguridad para la ejecución de los servlets, por ejemplo acceso al sistema de ficheros, o limitaciones en la creación de hilos.

El mecanismo de gestión del ciclo de vida de los servlets se basa en un contrato definido entre el contenedor y el servlet, es decir mediante una interfaz. Los servlets deben implementar la interfaz javax.servlet.Servlet para que el contenedor pueda comunicarse con ellos, el contenedor instancia e invoca el servlet por medio del API de reflexión. De esta manera el contenedor conoce y se asegura de que el componente que está instanciando es un Servlet y no otro. Aunque, cuando se habla de Servlets, se supone siempre una relación muy estrecha con la web, en realidad, el API servlet es independiente del protocolo, aunque la mayoría de implementaciones son para el protocolo HTTP, sería fácil implementar un Servlet FTP. El que un Servlet pueda ser independiente del protocolo es gracias a que la interfaz Servlet es muy genérica, de hecho el API proporciona una clase GenericServlet que dota de toda la funcionalidad para establecer peticiones y respuestas sobre cualquier protocolo, así mismo existe una clase HttpServlet que proporciona las mismas funcionalidades pero sobre protocolo http [31-25].

El ciclo de vida de un servlet viene definido en base a los métodos que proporciona el interfaz javax.servlet.Servlet, estos métodos son:

1 public void init(ServletConfig config) throws ServletException.

2 public void service(ServletRequest req, ServletResponse res) throws IOException, ServletException

3 public void destroy()

Con el método init(), la especificación se asegura de que el contenedor no llamará a ningún otro método antes de que este haya terminado, de esta manera sólo existirá un único proceso. Este método sólo es llamado una vez, normalmente bajo la primera petición del primer cliente, o bien precargando el servlet desde la consola de administración del servidor, o bien configurando este comportamiento en el descriptor de despliegue. El método init() típicamente es usado para la configuración de la aplicación, abrir recursos que debe de estar disponibles para el resto de la aplicación, como conexiones con bases de datos, acceso a un repositorio o sistema de ficheros, leer un archivo de configuración,

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

etc. El método service() lo invoca el contenedor de manera concurrente cada vez que recibe una petición para ese servlet. Antes de invocar a service, el contenedor construye dos objetos: Uno de tipo ServletRequest, que encapsula la petición, especialmente los parámetros. Otro de tipo ServletResponse, que encapsula la respuesta. Estos dos objetos son de tipo genérico y pueden establecer las peticiones y respuestas de cualquier protocolo, en el caso de http el API proporciona los métodos necesarios para establecer las cabeceras, códigos de estado y métodos de conexión adecuados. Por último el método destroy() es el último método que llamará el contenedor antes de destruir el servlet, las tareas que se realizan en este método son de liberación de recursos que ha usado el servlet, normalmente cerrará y limpiará aquellos recursos que se han abierto o usado en el init, aunque también en el service (conexiones con bases de datos, ficheros abierto), además de esta manera el contenedor se asegura de que no intentará destruir el servlet antes de que el método desrtoy haya terminado [36-39].

Conceptos básicos: Peticiones, Respuestas y Contexto.
La especificación Servlet define cuatro objetos básicos que son manejados por cualquier servlet:

Concepto	**Clase General**	**Clase HTTP**
Petición	javax.servlet.ServletRequest	javax.servlet.http.HttpServletRequest
Respuesta	javax.servlet.ServletResponse	javax.servlet.http.HttpServletResponse
Configuración	javax.servlet.ServletConfig	javax.servlet.ServletConfig
Contexto	javax.servlet.ServletContext	javax.servlet.ServletContext

Tanto el contexto, como la configuración del servlet son objetos independientes del protocolo, de ahí que no exista una implementación específica para http. El contexto define toda la configuración para una aplicación entera con todos sus componentes, mientras que la configuración define los parámetros de configuración para cada componente de manera individual. El contexto es el punto de contacto de los servlets con el sistema de archivos, ya que es el contenedor el que conoce el mapeo de directorios virtuales a directorios físicos. Así, el contexto proporciona métodos para recuperar recursos como para recuperar su tipo MIME:

1. public String getRealPath(String path)
2. public InputStream getResourceAsStream(String path)
3. public URL getResource(String path) throws MalformedURLException
4. public Set getResourcePaths(String path)
5. public String getMimeType(String file)

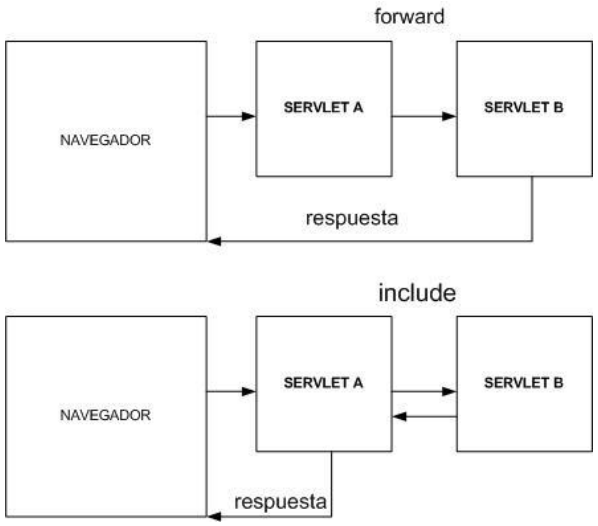
ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Las peticiones se encapsulan en el interfaz javax.servlet.ServletRequest, y ya que un servlet no tiene que estar ligado al protocolo http, y es independiente del protocolo usado, se pueden crear subinterfaces para protocolos específicos, siendo esto trabajo del proveedor del contenedor. Las peticiones encapsulan información sobre:

6. Parámetros de la petición.
7. Atributos.
8. Internacionalización.
9. El cuerpo de la petición.
10. El protocolo de la petición.
11. El servidor.
12. El cliente.
13. Redirección de la petición (Dispatchers).

La redirección de la petición puede venir por dos mecanismos, en ambos la petición original es pasada a otro componente y modificada para ser complementada y de esta manera producir la respuesta al cliente. Los dos mecanismos son forward o include. La diferencia entre uno u otro estriba en quién puede hacer la modificación de la petición y sobretodo quién envía la respuesta al cliente, tal y como se muestra en la siguiente figura:



54.Figura 16. Redirección de la petición.

Para dar soporte HTTP al API Servlet, éste incluye una serie de interfaces y clases, así, las peticiones HTTP están encapsuladas por la interfaz javax.servlet.http.HttpServletRequest que es subinterfaz de javax.servlet.ServletRequest. Este interfaz define métodos clasificables en los siguientes grupos:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

14. Seguridad
15. Cabeceras HTTP
16. Información sobre URL's
17. Métodos HTTP
18. Gestión de la sesión

Los parámetros de la petición se reciben desde el cliente en forma de cadenas de texto y son los que, habitualmente, dan información para la generación del contenido de la respuesta.
Los métodos de gestión de parámetros de la clase javax.servlet.ServletRequest son:

19. public String getParameter(String name) Devuelve el valor de un parámetro en función de su clave.
20. public Enumeration getParameterNames() Devuelve todos los nombres de los parámetros.
21. public String[] getParameterValues(String name) Devuelve todos los valores de un parámetro compuesto.
22. public Map getParameterMap()Devuelve un mapa con los pares clave/valor.

En el caso de las redirecciones, la misma petición puede pasar por varios servlets o JSPs, y el API nos permite modificar la petición enviada por el cliente añadiéndole una serie de atributos. Debe quedar claro que, mientras los parámetros llegan desde el cliente, los atributos se añaden durante la gestión de la petición, es decir, dentro del contenedor. Los métodos proporcionados son:

23. public Object getAttribute(String name) Devuelve el valor de un atributo en función de su clave.
24. public Enumeration getAttributeNames()Devuelve todos los nombres de los atributos.
25. public void setAttribute(String name, Object o) Añade un atributo a la petición.
26. public void removeAttribute(String name) Elimina un atributo de la petición.

Un mensaje de petición http, contiene un método de conexión y una serie de cabeceras que complementan al cuerpo de la petición. En algunos casos las peticiones pueden incluir más información de las que puden incluir las cabeceras, así algunas peticiones incluyen un cuerpo, por ejemplo haciendo uso del método de conexión PUT o POST. El API proporciona una serie de métodos para gestionar la información del cuerpo de la petición:

27. public String getCharacterEncoding()Devuelve el juego de caracteres del cuerpo de la petición.
28. public void setCharacterEncoding(String env) throws UnsupportedEncodingException Sobreescribe el juego de caracteres del cuerpo de la petición.
29. public int getContentLength()Devuelve la longitud total del cuerpo de la petición.
30. public String getContentType()Devuelve el tipo MIME del cuerpo de la petición.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

31. public ServletInputStream getInputStream() throws IOException Devuelve un InputStream para leer el cuerpo de la petición (binario).
32. public BufferedReader getReader() throws IOException Devuelve un Reader para leer el cuerpo de la petición (texto).

Para realizar la redirección de la petición se hace uso de los métodos forward(Request, Response) o include(Request, Response) (delegación/composición) que el API servlet proporciona un mecanismo basado en la interfaz RequestDispatcher, y desde la interfaz ServletRequest se puede recuperar un RequestDispatcher usando el método:
public RequestDispatcher getRequestDispatcher(String path).
Por ejemplo:

33. getServletContext().getRequestDispatcher("/pages/showBalance.jsp").forward(req,res).
34. getServletContext().getRequestDispatcher("/pages/navigation_bar.html").include(req,res);

Para dar soporte HTTP al API Servlet, éste incluye una serie de interfaces y clases, así, las peticiones HTTP están encapsuladas por la interfaz javax.servlet.http.HttpServletRequest (subinterfaz de javax.servlet.ServletRequest).Este interfaz define métodos clasificables en los siguientes grupos:

35. Seguridad
36. Cabeceras HTTP
37. Información sobre URL's
38. Métodos HTTP
39. Gestión de la sesión

El contenedor web proporciona mecanismos de gestión de la seguridad (autenticación y autorización) declarativos (mediante los descriptores de despliegue), pero es posible utilizar seguridad programática, para ello la interfaz HttpServletRequest declara los siguientes métodos:

40. public String getAuthType()
41. public String getRemoteUser()
42. public boolean isUserInRole(String role)
43. public Principal getUserPrincipal()

Así mismo se definen métodos para recuperar la metainformación de las cabeceras, como por ejemplo:

44. public long getDateHeader(String name)
45. public String getHeader(String name)
46. public Enumeration getHeaders(String name)
47. public Enumeration getHeaderNames()
48. public int getIntHeader(String name)

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Los servlets generan la respuesta a partir de un objeto que implementa la interfaz javax.servlet.ServletResponse que, básicamente, encapsula un OutputStream, sobre el que escribe el servlet. Los objetos ServletResponse son instanciados por el contenedor y pasados al servlet. El OutputStream del que depende esta interfaz se implementa usando un mecanismo de buffer intermedio, así, la interfaz proporciona métodos para controlar este buffer.
El cuerpo de la respuesta se escribe por medio de un stream que se apoya en un buffer. En el caso de ir a generar contenido dinámico se usará un javax.servlet.ServletOutputStream y en el caso de texto plano se usará un PrintWriter. Los métodos para recuperar, manejar y recuperar información del flujo de escritura son:

49. public ServletOutputStream getOutputStream() throws IOException

50. public PrintWriter getWriter() throws IOException

Estos dos métodos son mutuamente excluyentes, sí se ha invocado alguno de ellos una invocación sobre el otro lanzará IllegalStateException.

51. public void reset(). Vacía el stream, tanto el cuerpo como las cabeceras.

52. public boolean isCommitted(). Devuelve true si el buffer ha sido volcado (lo que implica que se han escrito las cabeceras y el cuerpo).

53. public void setBufferSize(int size) Establece el tamaño del buffer (el tamaño por defecto depende de la implementación).

54. public int getBufferSize()Devuelve el tamaño del buffer (o cero si no se está usando buffer).

55. public void flushBuffer() throws IOException Vuelca el buffer, es decir, escribe la respuesta (cuerpo y cabeceras), cualquier llamada a isCommited a partir de la invocación de este método devolverá true.

56. public void resetBuffer()Vacía el buffer, es decir, vacía el cuerpo, pero no las cabeceras.

El API sobrescribe el método service de la clase genérica, adaptándolo a los diferentes métodos de conexión en el protocolo http, así en lugar de disponer un único método service, existe un método por cada método de conexión siendo doGet, doPost, doPut, doDelete, doTrace, doOptions y doHead.

Listeners

La especificación Servlet añade una serie de escuchadores de eventos (siguiendo el modelo de eventos de Java) para dar la posibilidad al programador de reaccionar ante cambios de estado de la aplicación. Todos estos listeners siguen el mismo esquema (a excepción de HttpSessionBindingListener que fue incluido en la especificación 2.2 y tiene un comportamiento diferente).
Estos eventos pueden ser usados para conseguir persistencia (serializando objetos) y validación de estados.
Tales eventos de la aplicación necesitan ser declarados en el Deployment Descriptor, a continuación se presenta un resumen de los posibles escuchadores de eventos sobre los eventos desencadenados:
Eventos del contexto:

- ServletContextListener - ServletContextEvent

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- ServletContextAttributeListener – ServletContextAttributeEvent

Eventos de la Sesión:

- HttpSessionListener - HttpSessionEvent
- HttpSessionActivationListener - HttpSessionEvent
- HttpSessionAttributeListener - HttpSessionBindingEvent
- HttpSessionBindingListener – HttpSessionBindingEvent

Por ejemplo, Todos los escuchadores se declaran en el Deployment Descriptor mediante la siguiente estructura:

<listener><listener-class></listener-class></listener>

Es el contenedor, por medio de reflexión, el encargado de distinguir la clase del escuchador, es decir, no se indica en ningún momento qué tipo de escuchador se está registrando.
Una vez declaradas, podríamos utilizar por ejemplo la interfaz HttpSessionListener para trabajar con sesiones, Las implementaciones de esta interfaz reciben notificación de cambios en la lista de sesiones activas de la aplicación.
Los métodos por los que recibe notificación son:

- public void sessionCreated(HttpSessionEvent e) -> Se ha creado una nueva sesión.

- public void sessionDestroyed(HttpSessionEvent e) -> Se ha eliminado una sesión.

Y así podríamos utilizar esta información para cerrar recursos abiertos (por ejemplo conexiones con una base de datos, o ficheros para que no queden en un estado corrupto) cuando un usuario ha salido del sistema y ha cerrado su sesión.

Filtros

En versiones anteriores de la especificación se utilizaban métodos (ya obsoletos), para comunicar servlets entre sí, a partir de la especificación Servlet 2.3 se añadió a la especificación el mecanismo de filtros que complementa el mecanismo de dispatchers.
Permite el encadenamiento de filtros para la distribución de las tareas, así, un filtros validará, otro cifrará, y al final de la cadena, un servlet procesará, en lugar de tener un servlet monolítico que realice todas estas tareas.
Con los filtros el programador puede, por primera vez, interceptar el flujo de las peticiones en el contenedor de una manera portátil y estándar.
Los filtros capturan la petición y pueden cambiar el flujo de ésta (decidir a qué servlet dirigirla, rechazarla, encadenar una serie de servlets para procesar la petición secuencialmente, etc…).
Entre sus aplicaciones se incluyen:

- Autenticación y autorización, tanto en el contenedor como en recursos externos (sistemas legados).
- Sistemas de registro de eventos (log) y auditoria.
- Implementación de cachés de contenido generado.
- Control parametrizado de acceso (p.e. dependiendo de la hora/fecha).

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Transformaciones de la respuesta:
- Cifrado.
- Compresión.
- Transformaciones XSL/T.

Un filtro es una clase que implementa la interfaz javax.servlet.Filter que define los siguientes métodos:

- public void init(FilterConfig config) throws ServletException

Llamado cuando el contenedor inicializa el filtro.

- public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException

Llamado cuando el contenedor necesita que se ejecute el filtro.

- public void destroy()

Llamado cuando el contenedor va a descargar el filtro.
Un ejemplo:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Auditor implements Filter {
        private ServletContext ctx;
        public void init(FilterConfig fg) {ctx = fg.getServletContext();}
        public void doFilter(ServletRequest rq, ServletResponse rs, FilterChain
fc) throws ServletException, IOException {
                ctx.log("Conexión desde " + rq.getRemoteAddr());
                fc.doFilter(rq, rs);
        }
        public void destroy() {}
}
```

Como se puede observar, el filtro realizará una redirección mediante un dispatcher interno, pero en ningún momento se dice a qué clase hay que enviar la información en esta redirección. Esto se hace mediante el descriptor de despliegue, lo que facilita interconectar filtros sin modificar el código y por tanto recompilar como si lo hiciéramos con el RequestDispatcher tradicional.
La información en el descriptor de despliegue sería como la siguente:

```
<filter>
        <filter-name>Nombre del Filtro</filter-name>
        <filter-class>nombre.de.la.Clase</filter-class>
        <init-param> <!-- Opcional -->
          <param-name>nombre del parámetro</param-name>
                <param-value>valor del parámetro</param-value>
        </init-param>
</filter>

<filter-mapping>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
        <filter-name>Nombre del Filtro</filter-name>
        <url-mapping>/*</url-mapping>
</filter-mapping>
```

La especificación JSP.

Uno de los problemas heredados por los Servlets del modelo CGI clásico es que no existe separación de responsabilidades, un mismo componente contiene lógica de presentación y lógica de negocio, mezclando distintos lenguajes y tecnologías. Este hecho no permite la escalabilidad de las aplicaciones, ni la reutilización del código, obligando a recompilar los componentes de negocio por tareas, a veces sencillas, de modificación del interfaz, el mantenimiento de la aplicación puede llegar a ser un verdadero caos, siendo difícil aumentar las funcionalidades.
Bajo estas premisas, surge la especificación JSP, amparada por el éxito del modelo ASP de Microsoft viene a resolver estas carencias. Un JSP es un componente Java específicamente diseñado para la generación de la información al cliente de manera dinámica [40-42].

Un JSP es un documento de texto, normalmente escrito en código a html o XML con una serie de etiquetas dinámicas que permiten la ejecución de código Java, como documento de texto es fácilmente editable con cualquier editor de textos simple, esto permite una mayor sencillez a la hora de modificar el interfaz de usuario. El fichero debe tener la extensión .jsp, y dentro de un proyecto se instalarán como si fueran documentos html normales. Los JSP nos permiten eliminar el código html de los Servlets, separando la lógica de control de la vista, por tanto los Servlets quedarán como componentes de control. La parte estática de un JSP vendrá definida por las etiquetas propias del lenguaje (ya sea html o XML), y la parte dinámica puede venir definida por dos tipos de etiquetas, por un lado el formato clásico y por otra mediante un formato XML.

1. <% %>: formato clásico
2. <jsp: > </jsp>: formato XML

```
<HTML>
        <HEAD>
                <TITLE>Ejemplo Hola mundo  en una JSP</TITLE>
        </HEAD>
          <BODY>
            <% out.println("Hola mundo desde una JSP"); %>
          </BODY>
</HTML>
```

55.Figura 17. JSP de Ejemplo.

Por tanto, los JSP son vistos como plantillas que necesitan de una interpretación en el servidor para producir la respuesta dinámica. Los JSP sufren una compilación en el servidor, el contenedor detectará que es un tipo de componente, y necesita ser compilado antes de devolver la respuesta al cliente. La compilación que sufre un JSP, internamente es a un Servlet, por lo tanto comparten el mismo API, los mismos objetos petición y respuesta, también dispondrá de un objeto contexto y un objeto de configuración, y también tienen un ciclo de vida al igual que los Servlets manejando un método inició, un método de servicio, y un método de destrucción. Todo esto permite al contenedor configurar los proyectos web por igual, y los componentes pueden trabajar juntos compartiendo datos mediante

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

sesiones, o redireccionando la petición. El JSP contiene una serie de elementos para su trabajo, como son:

1. Expresiones de la forma **<%= expresión %>** que son evaluadas e insertadas en la salida (equivalentes a un out.println).
2. Scriptlets de la forma **<% codido %>** que se insertan dentro del método service() del servlet, usadas para la creación de algoritmos.
3. Declaraciones de la forma **<%! codigo %>** que se insertan en el cuerpo de la clase del servlet, y permiten la declaración de variables y métodos.
4. Directivas de la forma **<%@ directive atributo1="valor1"... atributoN="valorN" %>** que afectan a la estructura general del servlet generado, como por ejemplo importar librerías, establecer el Content-Type, establecer el tamaño y comportamiento del buffer de salida, etc.

En el siguiente ejemplo se muestra el uso de las etiquetas en un JSP, en concreto se usan scriptles y expresiones y se muestra como es posible usar los mismos objetos proporcionados por el API java como es el objeto Date, y recuperar parámetros proporcionados por el cliente mediante el objeto HttpServletRequest, además se muestra como es posible escribir en la salida estándar mediante el método println de igual forma que hace una expresión.

```
<HTML>
<HEAD>
<TITLE>Ejemplo de JSP</TITLE>
</HEAD>
<% String bgColor = request.getParameter("bgColor");
            boolean hasExplicitColor;
            if (bgColor != null) {
              hasExplicitColor = true;
            } else {
              hasExplicitColor = false;
              bgColor = "WHITE";
} %>
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">El  color de Fondo es: <%= bgcolor %> </H2>
<H2>Ejemplos de expresiones JSP</H2>
<UL>
  <LI>Hora actual en el servidor: <%= new java.util.Date() %>
  <LI>Nombre del host: <%= request.getRemoteHost() %>
   <LI>Valor del parametro testParam:   <% out.println( request.getParameter("testParam") );%>
</UL>
</BODY>
</HTML>
```

56.Figura 18. Ejemplo de uso de etiquetas JSP.

Sin embargo en este ejemplo se hace uso del objeto request y del objeto out sin ningún tipo de declaración previa, esto es debido a que la especificación JSP adopta una serie de objetos implícitos que permiten el trabajo de la misma forma que se hacia con los servlets, como son:

1. request.
2. response.
3. out.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

4. session.
5. application.
6. config.
7. pageContext.
8. page.
9. exception

El uso de JSP es útil para no mezclar lógica de negocio y lógica de presentación, sin embargo es muy fácil ligar de manera dura a un JSP con la parte de negocio, debido a que comparte el mismo API que un Servlet, el hecho de que se compile como tal y funcione de la misma forma, permite a un JSP hacer exactamente las mismas tareas que un Servlet. Sin embargo, no hay que caer en esa tentación, el JSP debería estar destinado al interfaz de usuario, no a la lógica de negocio, ni enlazarse directamente con los datos. Para desacoplar a los JSP del resto de capas de la aplicación, existen una serie de componentes intermedios que fueron mencionados al principio del documento, cuando se hacía una evolución de las arquitecturas web, estos componentes albergan en su interior lógica de negocio y acceso a datos, lo que les permite ser reutilizables, de esta manera la aplicación es más mantenible y escalable. Típicamente estos componentes son JavaBeans, Taglisbs y EJBs [43-45].

JavaBeans y JSP

La acción <jsp:useBean> nos permite localizar o instanciar un JavaBean en la página JSP. La sintaxis más simple para especificar que se debería usar un Bean es:

<jsp:useBean id="nombre" class="paquete.clase" />

Con esto conseguimos localizar o instanciar un objeto, de la clase especificada por "class", y enlazarlo con una variable, con el nombre especificado en "id".

También se puede especificar un atributo "scope" que hace que el bean se asocie con más de una página. En este caso, es útil obtener referencias a los beans existentes, y la acción jsp:useBean especifica que se instanciará un nuevo objeto si no existe uno con el mismo nombre y ámbito.

Como un JavaBean es una clase cuyos métodos son codificados como propiedades (get, set), esta forma de trabajo nos permite acceder a dichas propiedades mediante setProperty y getProperty.

Usamos **jsp:setProperty** para establecer los valores de las propiedades de los beans que se han referenciado anteriormente con la acción jsp:useBean. Además valor de id en jsp:useBean debe coincidir con el valor de name en jsp:setProperty.

La acción **jsp:getProperty** obtiene el valor de una propiedad de un bean, usando el método getter del bean, e inserta su valor en la respuesta.

Antes de usar esta acción debe aparecer una acción jsp:useBean para instanciar o localizar el bean.

Librerías de etiquetas: Taglibs

Gracias a la directiva taglib podemos definir nuestros propios tags JSP.

Para definir un tag necesitamos definir 4 componentes:

- Una clase java que defina el comportamiento del tag.
- Un fichero TLD (Tag library descriptor) para hacer visible la clase en el servidor.
- Un fichero web.xml para hacer visible el tag en el servidor.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Un fichero JSP que use el tag

La sintaxis completa de la directiva taglib es:

<%@ taglib uri="URIForLibrary" prefix="tagPrefix" %>

El atributo uri define donde se localiza el fichero TLD, y puede ser un URL, un URN o un PATH absoluto o relativo. Si la URI es una URL, entonces el TLD es localizado por medio del mapping definido dentro del fichero web.xml. Si la URI es un path, entonces es interpretado como un acceso relativo a la raíz de la aplicación y debería resolverse en un fichero TLD directamente, o un fichero .jar que contiene el fichero TLD.

De esta forma podemos tener un conjunto de librerías de etiquetas muy útiles para diseñadores de interfaz de usuario que necesiten realizar operaciones complejas (como extraer datos de una base de datos) sin conocimientos en java, o bien para personal externo a nuestras aplicaciones que necesiten llevar a cabo dichas operaciones, pero no queremos que conozcan nada de nuestros sistemas. Así nos aseguramos de que todo el mundo utilizará los accesos a las bases de datos sin incorporar código propietario, por ejemplo.

Como conclusión las librerías de etiquetas facilitan el desarrollo de páginas jsp reduciendo su complejidad y número de líneas de código y facilitan la seguridad, la reutilización y el mantenimiento.

A continuación, se muestra un ejemplo de uso de una taglib:

Fichero Hola.jsp

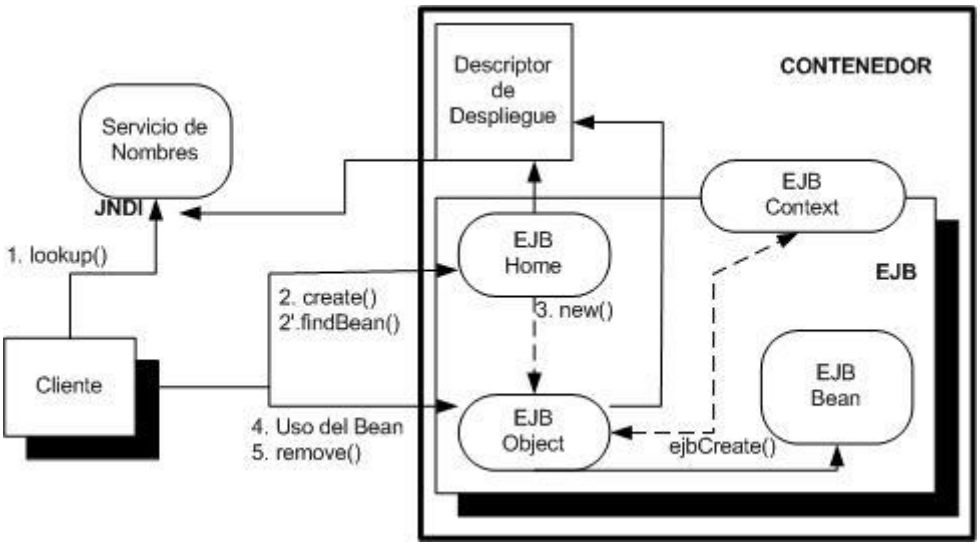
```
<HTML>
<HEAD>

<%@ taglib uri="Hola.tld" prefix="test" %>

<TITLE><test:HolaMundo /></TITLE>

<BODY>

<H1> <test:HolaMundo /></H1>

</BODY>
</HTML>
```

Programación distribuida: el modelo EJB.

La evolución de las arquitecturas cliente-servidor tradicionales, siempre han buscado el mismo fundamento, escalabilidad de las aplicaciones, reutilización de las aplicaciones a modo de servicios y minimizar en lo posible el impacto de un cambio de plataforma, nuevas funcionalidades, y cambios en la plataforma cliente. Hasta ahora la invocación remota de procedimientos (RPC), o el estándar

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

CORBA han proporcionado el sustento para la computación distribuida y dentro de la especificación J2EE aparece el modelo de componentes distribuidos denominados Enterprise JavaBeans (EJB).
La especificación J2EE también proporciona un modelo de contenedor para los EJBs con el fin de proporcionar una arquitectura estándar para la construcción de aplicaciones empresariales distribuidas basadas en componentes, guiar todo el ciclo de vida de la aplicación: desarrollo, despliegue y gestión en tiempo de ejecución, proporcionar portabilidad entre plataformas.(Write Once, Run Anywere) y ocultar al desarrollador de los detalles de bajo nivel relacionados con la seguridad, transacciones, distribución de objetos, concurrencia, conexiones con repositorios, etc.
En un modelo distribuido, los servicios no son accesibles de manera directa, son independientes del cliente que los usa y por tanto no tienen por qué estar en la misma máquina, ni el mismo segmento de red en el que se encuentra al cliente. De esta manera, es necesario proporcionar un sistema mediante el cual el servidor publique sus servicios y el cliente pueda buscarlos de manera remota antes de enlazarse con el servidor. El servidor publica en un servidor de nombres los detalles del servicio, siendo este una estructura jerárquica en forma de árbol, y el cliente navega a través del servidor de nombres buscando el servicio que necesita, una vez encontrado obtendrá toda la información necesaria para hacer uso de ese servicio [46].
En el modelo EJB el cliente nunca accederá directamente al componente que implementa el servicio, la localización, instanciación y uso del servicio se realiza por medio de una serie de interfaces, que rutarán entre el cliente y el servidor final, tal y como muestra la siguiente imagen:



El servicio implementado con la lógica de negocio viene representado por el EJB Bean, para acceder a él, el cliente tendrá que localizar su referencia en un servicio de nombres, al que accederá mediante JNDI, donde localizará el interfaz Home, que es el que incorpora los métodos necesarios para la búsqueda e instanciación del servicio. Una vez creado, el interfaz Home, devolverá un interfaz Object al cliente, este interfaz es el que tiene una descripción de los métodos de negocio que el cliente puede usar, toda llamada a estos métodos producirá una delegación a los métodos reales proporcionados por el EJB Bean. De esta manera el cliente nunca accede directamente a los métodos proporcionados por

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

el servicio. La configuración del comportamiento del componente, ciclo de vida, transacciones, seguridad, etc, viene definida de manera declarativa en el descriptor de despliegue que es gestionado por el contenedor.

Modelo de componentes.

Existen diversos componentes dependiendo del tipo de servicio que queramos proporcionar, se establecen tres tipos:

1. Componentes de Entidad (Entity Beans).
2. Componentes de Sesión (Session Beans)
3. Componentes de Mensajería (Message Drive Beans).

Componentes de Entidad.

Este tipo de componentes representan datos de la base de datos vistos como objetos, es decir son persistentes. Disponen de un acceso compartido ya que varios clientes pueden acceder a los mismos datos, y por lo tanto su ciclo de vida es largo, durarán tanto como duren los datos en la base de datos y como consecuencia sobreviven a las caídas de sistema. Dentro de este tipo de componentes hay dos variantes:

1. Container Manager Persistence (CMP).

2. Bean Manager Persistence (BMP).

El tipo CMP es gestionado íntegramente por el contenedor, realizando las operaciones necesarias para acceder a los datos de la base de datos, mientras que el tipo BMP es gestionado prácticamente por el desarrollador del componente. Todos los EJBs disponen de un ciclo de vida gestionado por el contenedor, la diferencia entre el tipo CMP y el BMP reside en que en el primero es el contenedor el que realiza todo el trabajo de acceso a datos, bloqueo, transaccionalidad, mapeo objeto-.relacional, etc. Mientras que en el segundo caso es cuestión del desarrollador proporcionar los mecanismos necesarios para realizar esas tareas. El tipo BMP es bastante más trabajoso que el CMP, pero proporciona más control y se suele utilizar cuando el modelo CMP no proporciona todo el control que requiere la aplicación.

Componentes de Sesión.

Estos componentes se ejecutan ligados a un cliente, por tanto, no son compartidos entre clientes, pueden ser transaccionales y aunque no representan datos en la base de datos pueden acceder y actualizar dichos datos, pero no es su cometido. Al no representar datos y estar ligados a un cliente, tienen una vida relativamente corta, tanto como dure la conversación con el cliente. No sobreviven a las caídas del sistema. Son la representación lógica del cliente en el servidor, contienen información específica del cliente y representan los procesos de negocio de la aplicación. Su uso suele ser para

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

realizar las operaciones de negocio, establecer flujos de peticiones y ocultar los detalles de implementación de los servicios proporcionados por los EJB de entidad a los clientes, accediendo por tanto los EJBs de sesión a los de entidad y haciendo de cliente de estos.
Dentro de este tipo de componentes hay dos variaciones:

1. Sesión sin estado (Session Beans Stateless).
2. Sesión con estado (Session Beans Stateful).

Los EJB de sesión sin estado no mantienen el estado conversacional con un cliente dado, son ligeros y no guardan ningún dato que pueda identificar al cliente, al revés que los EJB de sesión con estado cuyo cometido es proporcionar los mecanismos necesarios para mantener la conexión con el cliente hasta que o bien este o el servidor decidan cancelar la comunicación, este tipo de componentes suele llevar a cabo una serie de funcionalidades añadidas, de gestión y mantenimiento que hace que sean más pesados que sus compañeros, normalmente con operaciones de escritura en disco (serialización de objetos), que en algunos sistemas pueden producir efectos adversos de ralentización o en clustering y balanceo de carga.

Componentes de mensajería.

Los MDB aparecieron a partir de la especificación EJB 2.0, debido a la carencia de un modelo asíncrono. Utilizan JMS para encaminar los mensajes del cliente, pueden ser transaccionales, y aunque su cometido no es representar datos, pueden acceder y actualizar dichos datos. Su finalidad es prácticamente la misma que la de los EJB de Sesión, modelar la lógica de negocio de la aplicación, pero desde un punto de vista asíncrono, por lo tanto su vida es relativamente corta y tampoco sobreviven a las caídas del sistema.

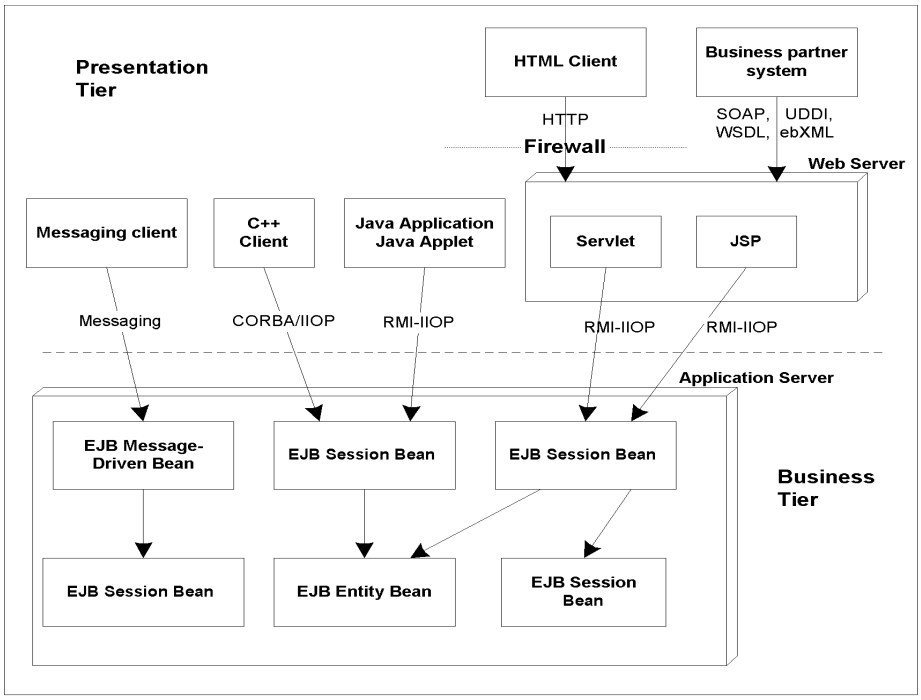
Novedades

Las novedades más importantes surgen a partir de la versión EJB 2.0, donde como ya se ha comentado se proporciona el soporte a la comunicación asíncrona mediante los MDBs, pero también se proporciona la posibilidad de publicar de manera local los interfaces de conexión de los componentes en lugar de manera remota como era antes obligado, lo que afectaba al rendimiento en algunas aplicaciones, sobretodo cuando toda la infraestructura se gestionaba en la misma máquina. La tercera novedad es la posibilidad de publicar los servicios usando la infraestructura de servicios web mediante XML, así usar el descubrimiento mediante UDDI y uso de los interfaces mediante WSDL usando SOAP como protocolo de transporte [47].

Uniéndolo todo. El servidor de aplicaciones

La especificación J2EE define un modelo común de desarrollo e implementación de aplicaciones de manera que estas sean portables y ejecutables entre distintas herramientas que cumplan la especificación con mínimo impacto, para ello define su modelo de contendor y el contrato con los componentes desplegados en el. Una arquitectura basada en J2EE da soporte para múltiples tipos de aplicaciones tal y como se muestra en la figura siguiente:

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241



57.

58.Figura 19. Tipos de aplicaciones J2EE.

En la figura se aprecia los dos tipos de contenedores que forma parte de un servidor de aplicaciones en contenedor web y el contenedor EJB, dentro de la parte EJB aparecen los componentes comentados anteriormente y como los EJB de Sesión y MDBs hacen de clientes de los de Entidad ocultando la implementación hacia el cliente final, el contenedor también da soporte a clientes asíncronos mediante los MDBs y a aplicaciones que no sean Java, mediante el uso de los APIs proporcionados para comunicaciones con aplicaciones CORBA. Asimismo aparece una caja que simula aplicaciones Java que no son web, estas aplicaciones pueden acceder a los servicios implementados con EJBs haciendo uso de RMI, con lo que las aplicaciones en tecnología J2EE no tienen que ir orientadas exclusivamente hacia el mundo web, para el cual también hay soporte mediante el contenedor web, a través de los componentes ya mencionados (Servlets y JSPS). Si se usa una arquitectura web con un cliente ligero basado en html puede acceder a la aplicación por este camino, internamente los servlets harán de clientes de los EJBs usando RMI. La incorporación de los Servicios Web a este tipo de arquitecturas proporciona un marco idóneo, ya que la integración de aplicaciones no-Java pueden ser hechas usando el protocolo de transporte http y la infraestructura web de manera más sencilla que implementar la comunicación a través de un ORB de CORBA, o usando Sockets o RMI.

ISBN: 978-84-9012-861-9

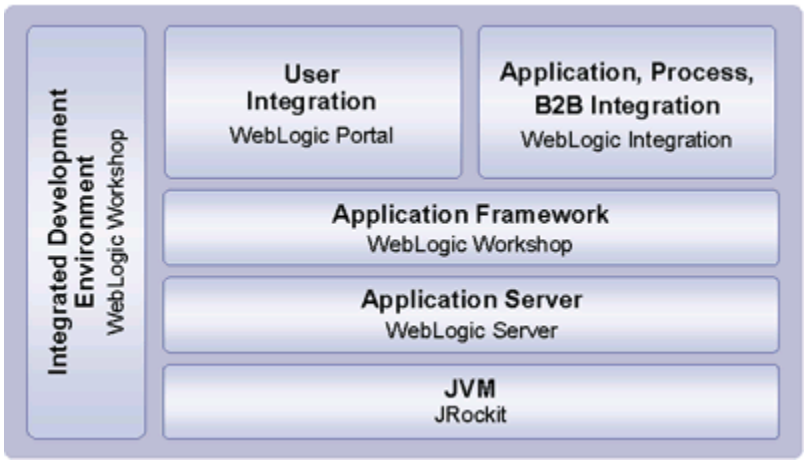
Colección Aquilafuente: AQ-241

La plataforma BEA WebLogic

La plataforma WebLogic constituye un conjunto de productos relacionados orientados a diferentes necesidades de desarrollo o producción. En su actual versión, la plataforma incluye los siguientes elementos:

1. *WebLogic Server*: Un servidor de aplicaciones basado en J2EE, junto a sus herramientas de desarrollo y administración.
2. *WebLogic Workshop*: Un entorno de desarrollo visual para el desarrollo de aplicaciones con la plataforma.
3. *WebLogic Integration*: Una solución de integración de aplicaciones especialmente indicada para el B2B.
4. *WebLogic Portal*: Un marco (*framework*) para el desarrollo de portales sobre la plataforma, incluyendo administración, personalización y gestión de contenidos.
5. *WebLogic JRockit*: Una máquina virtual Java optimizada para un mayor rendimiento y escalabilidad en el servidor.

El siguiente esquema muestra la relación entre los componentes mencionados.



59.Figura 20: Componentes BEA Weblogic

De estos componentes, aquí nos centramos en WebLogic Portal, que proporciona el soporte fundamental para el desarrollo de aplicaciones B2C.

Arquitectura de la plataforma WebLogic

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

La herramienta de configuración de WebLogic nos permite crear configuraciones para el desarrollo de diferentes tipos, teniendo habilitados unos servicios u otros. La Figura 1 nos muestra las configuraciones predefinidas que podemos seleccionar.

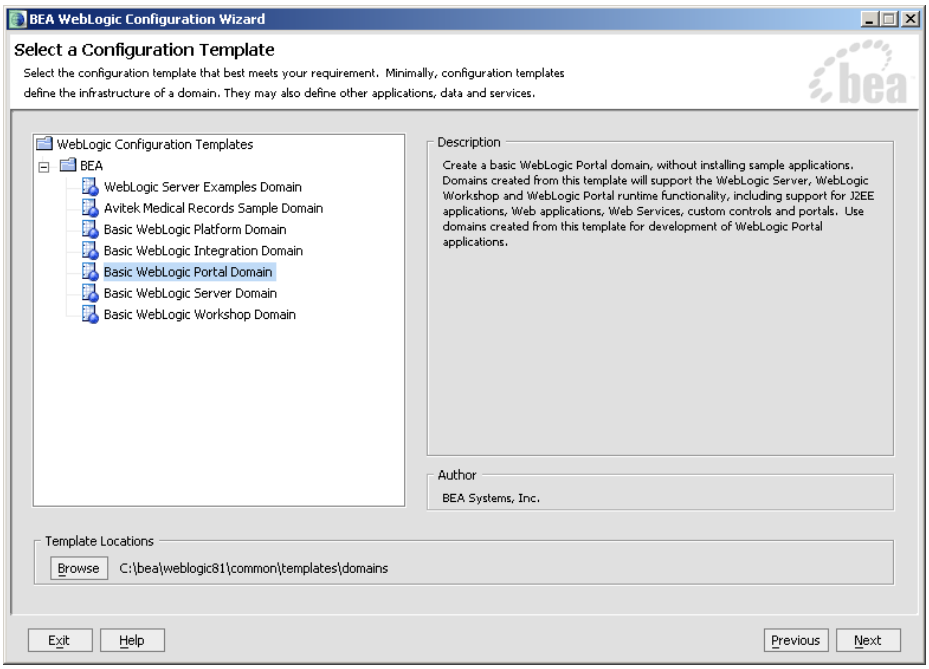


Figura 2: Configuración inicial de WebLogic platform

Al crear la configuración, se nos preguntará por una cuenta (usuario y password) para comenzar la administración posteriormente, y se generarán los ficheros iniciales de la aplicación en el directorio indicado, por ejemplo, en C:\bea\user_projects\domains\portalDomain. En ese directorio se encuentra un *script* de inicialización startWebLogic.cmd que utilizaremos para arrancar el servidor de administración.

Las instalaciones de WebLogic tienen como unidad administrativa el **dominio**, que representa un conjunto de recursos. Estos recursos pueden incluir más de un servidor WebLogic, y permite configuraciones en clúster de servidores, pero siempre hay uno de los servidores que funciona como Servidor Administrativo, siendo los demás "servidores gestionados" (*managed*). De esta manera se permite la *partición* de las aplicaciones, de manera que diferentes funcionalidades se asignen a diferentes servidores, y también la provisión de mecanismos de redundancia para mejorar la tolerancia a fallos de la instalación. De hecho, es recomendable que las aplicaciones en funcionamiento se asignen a servidores gestionados, de modo que la máquina de administración se utilice solamente para soportar tareas de administración.

La configuración del dominio se guarda en un fichero config.xml en la máquina que alberga al servidor de administración. La *Consola de Administración del Sistema* (CAS) puede utilizarse para crear

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

y modificar configuraciones mediante una interfaz gráfica de usuario. La Figura 2 muestra la apariencia de la consola de administración (accesible por HTTP a través del puerto de administración, por defecto el 7001), concretamente la página de administración de servidores dentro del dominio.

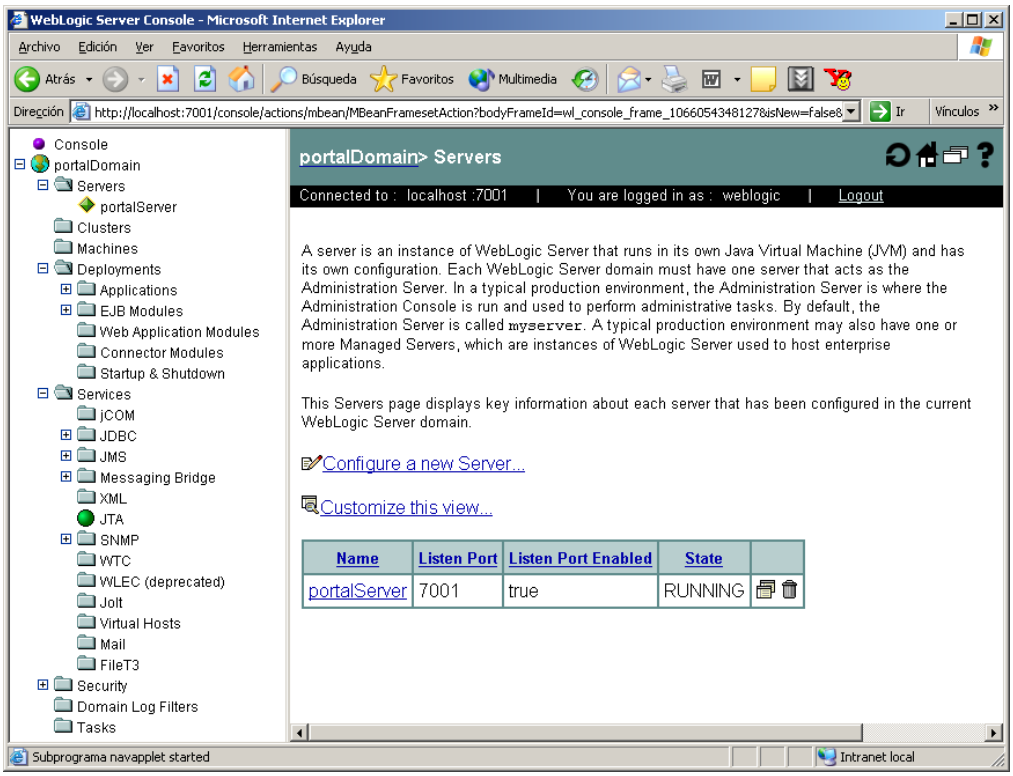


Figura 3: Consola de Administración de WebLogic platform

En la Figura 2 se pueden apreciar las diferentes opciones de administración, incluyendo clusters, instalación de aplicaciones o componentes (*deployments*), y otros servicios como controladores JDBC para acceso a bases de datos, o acceso a colas de mensajes JMS. También se puede configurar todo lo relativo a la seguridad, y permite el seguimiento de tareas (*tasks*) de administración que tardan un cierto tiempo en ejecutarse.

Desarrollo de portales con la plataforma WebLogic

Una vez creado un dominio, se pueden crear aplicaciones dentro de él. Para ello, se puede utilizar el editor integrado WebLogic Workshop. Cuando se utiliza *File|New|Application* tenemos que decidir el tipo de aplicación que queremos crear, como se muestra en la Figura 3.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

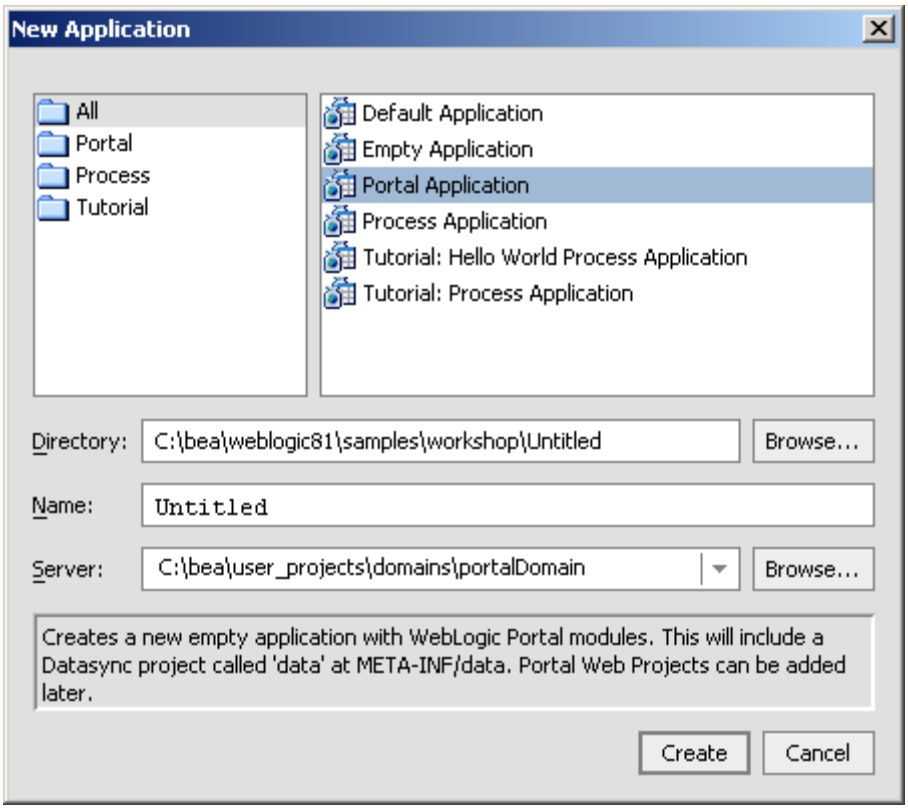


Figura 4: Creación de aplicaciones con WebLogic Workshop

La administración de portales en WebLogic se realiza con otra herramienta administrativa diferente a la Consola de Administración. Esta herramienta está accesible a través de la URL http://localhost:7001/Aplicacion1Admin, donde Aplicacion1 debe cambiarse por el nombre de la aplicación concreta que queramos administrar.

Examinemos a continuación un *portlet* muy sencillo, que implementa un diccionario (puede encontrarse en los tutoriales del producto):

```
<%@ taglib uri="render.tld" prefix="render" %>
<%-- ------------------------------------------------------------ --%>
<%-- Local variables                                              --%>
<%-- ------------------------------------------------------------ --%>
<%
   // url prefix of site
   String urlValue = "http://www.m-w.com/cgi-bin/dictionary?";

   // name of query parameter for the site
   String symbolParameterName = "va";

   // attributes of pop up windows
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
   String windowAttributes = "toolbar=no,alwaysRaise=yes,resizable=yes,scroll-
bars=yes,width=600,height=300";
%>
<%-- ------------------------------------------------------------ --%>
<%-- Portlet HTML content                                         --%>
<%-- ------------------------------------------------------------ --%>
<form method="post" action="<%=urlValue%>" onsubmit=" return <render:encodeName
name="DictionaryResultWindow"/>(this)">
   <table border="0" align="center">
      <tr>
         <td align="center">Enter the word you want to look up:</td>
      </tr>
      <tr>
         <td align="center">
            <input type="text" name="<%=symbolParameterName%>" size="15" maxlength="50" >
            <input type="hidden" name="lang" value="en">
            <input type="submit" name="wordSearchButton" value="Look up">
         </td>
      </tr>
   </table>
</form>

<script language="javascript">

/*
 * this is the local form handling.
 * since the actual search engine URL is not known till submit,
 * this intercepts the submit data and sends the full search URL
 * to the pop up window
 *
 * the function name must be unique to avoid conflict with other portlets
 */
function <render:encodeName name="DictionaryResultWindow"/>( form )
{
   // if browser is Netscape 4.x, let the from open the new
   // window instead of the script
   var appName = navigator.userAgent.toLowerCase();
   var isNetscape = ((appName.indexOf("mozilla") != -1) &&
                     (appName.indexOf("compat") == -1));
   var majorVersion = parseInt(navigator.appVersion);
   if (isNetscape && (majorVersion == 4)) return true;

   var urlPrefix = "<%=urlValue%>";
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
   var searchValue = form.elements["<%=symbolParameterName%>"].value;
   var fullUrl = urlPrefix + "<%=symbolParameterName%>=" +searchValue;

   // create initial window if it doesn't already exists
   window.open(fullUrl, "<render:encodeName name="result_window"/>", "<%= windowAttributes
%>");

   // never submit the form
   return false;
}

</script>
```

En el código anterior podemos apreciar que el *portlet* no es otra cosa que un fichero JSP con sus partes de código HTML, de gestión en cliente (javascript) y de código específico del *portlet*. Por tanto, la idea de portlet no es más que la de reutilizar fragmentos típicos de código Web, de manera que al registrarlos y definir su configuración y parámetros dentro de la plataforma WebLogic, se podrán reutilizar en diferentes portales.

La plataforma ATG

La plataforma ATG proporciona dos productos por separado que pueden utilizarse como infraestructura y soporte al desarrollo de aplicaciones B2C:

1. *Dynamo Application Server* (DAS) es un servidor de aplicaciones basado en J2EE.
2. *ATG* es un conjunto de productos que proporcionan soporte a la construcción de portales.

La versión actual de ATG es la 6.1 y es independiente del servidor de aplicaciones sobre la que se instala. Actualmente, soporta tanto DAS que es del mismo fabricante, como otros productos de fabricantes diferentes, concretamente, soporta los servidores J2EE de *WebLogic* y el *WebSphere* de IBM. De esta manera, lo propio de la gestión y desarrollo de portales en ATG permite utilizar un servidor de aplicaciones de otro fabricante.

Arquitectura de la plataforma ATG

Los componentes básicos de la plataforma ATG se muestran en el siguiente diagrama.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

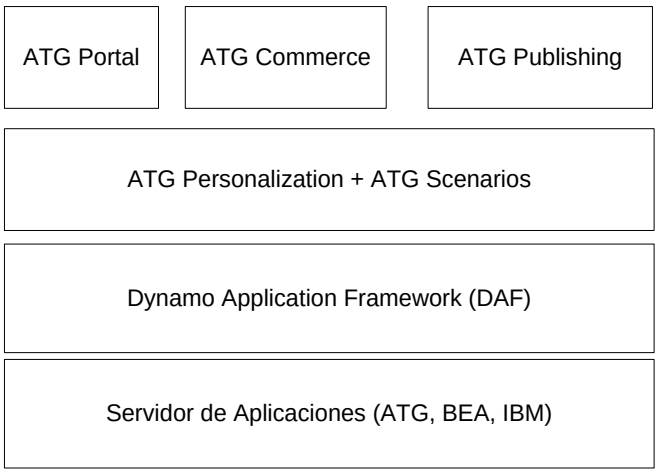


Figura 5: Componentes básicos de ATG.

Los componentes de ATG son, por tanto:

1. El servidor de aplicaciones, de entre los soportados por ATG.
2. Un marco de desarrollo básico, denominado DAF, que proporciona facilidades para el desarrollo e instalación de aplicaciones.
3. Dos módulos de personalización integrados. ATG Personalization proporciona un sistema de personalización basado en reglas que permite definir categorías de usuarios y de contenidos, y utilizar reglas para asociar contenidos personalizados a los usuarios. ATG Scenarios permite programar escenarios de interacción con los clientes, de modo que el sistema puede hacer campañas promocionales automatizadas y responder automáticamente a los usuarios de acuerdo al comportamiento de estos.
4. Tres productos basados en las capas anteriores. ATG Portal está orientado a construir portales para audiencias concretas mediante la reutilización de componentes. ATG Commerce proporciona servicios básicos de gestión de transacciones de compra y gestión de catálogo. Finalmente, ATG Publishing proporciona servicios de gestión de contenidos, incluyendo la posibilidad de definir y mantener flujos de trabajo (*workflows*).

La aplicación *ATG Control Center* (ACC) es el entorno de administración y desarrollo integrado para los elementos que acabamos de describir.

. Desarrollo de portales con la plataforma ATG

Los portales en ATG son colecciones de contenidos, servicios y usuarios. Los usuarios se pueden organizar en **comunidades.** ATG utiliza también el concepto de *Gear*, que viene a ser una utilidad o un área de contenidos de una página dentro de un portal (por ejemplo, un área de noticias, un foro, etc.). Son el equivalente al concepto de *Portlet* configurable y reutilizable de WebLogic Portal. Desde el punto de vista técnico, no son otra cosa más que un conjunto de ficheros JSP, clases Java y acceso

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

a servicio y datos que se pueden incluir y configurar en cualquier página del portal. Este es el concepto más relevante del desarrollo de portales en ATG.

La siguiente figura nos muestra un ejemplo ilustrativo de Gear soportando un foro de discusión, que se proporciona en la documentación de ATG, mostrándose en modo compartido, es decir, con otras instancias en la página.

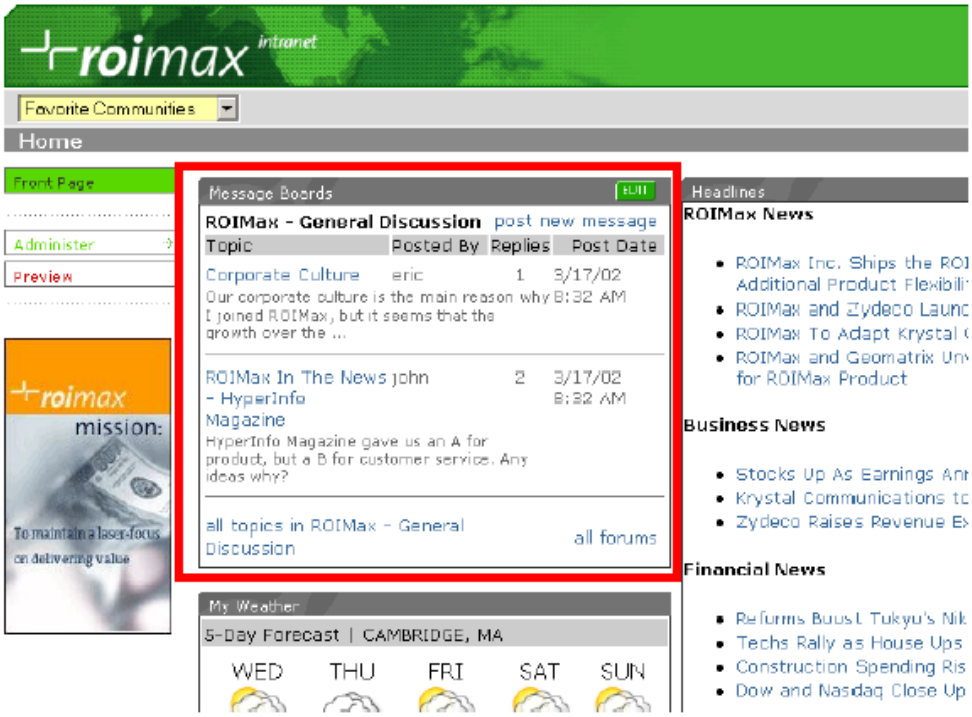


Figura 6: Gears en ATG.

Se pueden desarrollar *templates* de Gears, de modo que su registro posterior en el framework ATG los hace disponibles como componentes a cualquier portal. También se tiene la opción de diseñar para un mismo gear más de un perfil de visualización, de modo que se pueda mostrar, por ejemplo, mediante HTML y mediante WML en el caso de dispositivos móviles.

Veamos como ejemplo ilustrativo el código de la versión HTML de un *gear* mínimo:

```
<%@ page import="java.io.*,java.util.*,atg.portal.servlet.*,atg.portal.framework.*" %>
<%@ taglib uri="/jakarta-i18n-1.0" prefix="i18n" %>

<%
  GearServletResponse gearServletResponse =
    (GearServletResponse)request.getAttribute(Attribute.GEARSERVLETRESPONSE);
  GearServletRequest gearServletRequest =
    (GearServletRequest)request.getAttribute(Attribute.GEARSERVLETREQUEST);
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
  GearContextImpl gearContext  = null;
  if((gearServletResponse != null) &&
     (gearServletRequest != null))
    gearContext = new GearContextImpl(gearServletRequest);
%>

<i18n:bundle baseName="atg.gears.helloworld.helloworld" localeAttribute="userLocale" changeResponseLocale="false"
/>

<h3><i18n:message key="helloworld-shared"/></h3>
<%
  //Full Mode
  gearContext.setDisplayMode(DisplayMode.FULL);
%>
<p>
 <i18n:message key="clickhere">
    <% String newFullGearUrlString = "<a href=\"" + gearServletResponse.encodeGearURL(gearServletRequest.getPor-
talRequestURI(),gearContext) + "\">"; %>
    <i18n:messageArg value="<%= newFullGearUrlString %>"/>
    <i18n:messageArg value="</a>"/>
 </i18n:message>
</p>

<h3><i18n:message key="gear-environment"/></h3>
<ul>
 <li><i18n:message key="orig-request-uri-label"/><%= gearServletRequest.getPortalRequestURI() %></li>
 <li><i18n:message key="community-label"/><%= gearServletRequest.getCommunity() %></li>
 <li><i18n:message key="page-label"/><%= gearServletRequest.getPage() %></li>
 <li><i18n:message key="displaymode-label"/><%= gearServletRequest.getDisplayMode() %></li>
 <li><i18n:message key="gear-label"/><%= gearServletRequest.getGear() %></li>
</ul>
```

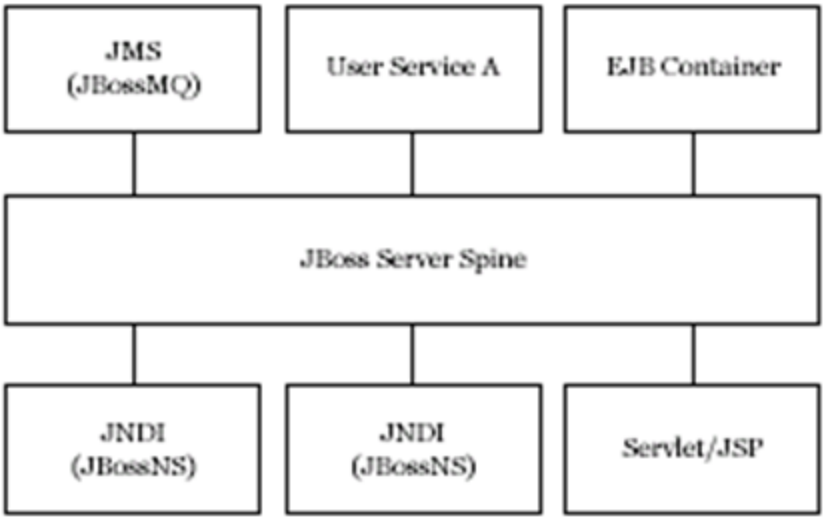
Como puede verse en el ejemplo anterior, tenemos simplemente un código JSP que invoca a ciertas funciones especiales, y muestra en la página información sobre sí mismo. A través de la instancia gearContext se puede controlar el comportamiento del mismo. El *gear* está además compuesto por otros ficheros de configuración, y posiblemente otros ficheros para versiones en otros lenguajes como WML o cHTML.

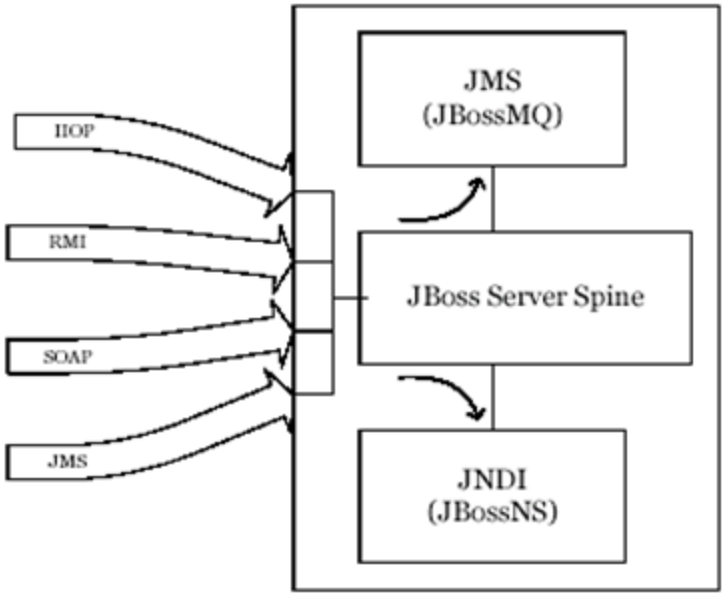
Jboss

JBoss es un servidor de aplicaciones J2EE de código abierto implementado en Java puro. Al estar basado en Java, JBoss puede ser utilizado en cualquier sistema operativo que lo soporte. Los principales desarrolladores trabajan para una empresa de servicios, JBoss Inc., adquirida por Red Hat en Abril del 2006, fundada por Marc Fleury, el creador de la primera versión de JBoss. El proyecto está apoyado por una red mundial de colaboradores. Los ingresos de la empresa están basados en un modelo de negocio de servicios.

JBoss implementa todo el paquete de servicios de J2EE. A modo de ejemplo, Los Sims online, utilizan JBoss para sus juegos multiusuario.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc





Jboss Portal

Es una plataforma de código abierto para albergar y servir un interfaz de portales Web, publicando y gestionando el contenido así como adaptando el aspecto de la presentación.

Como características principales categorizadas cabe destacar:

Tecnología y arquitectura

- JEMS: hace uso del potencial de JBoss Enterprise Middleware Services : JBoss Application Server, * JBoss Cache, Jgroups e Hibernate.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- DB Agnóstico: funciona con cualquier SGBD soportado por Hibernate
- SSO/LDAP: hace uso de las soluciones de single sign on (SSO) de Tomcat y JBoss
- Autenticación JAAS: módulos de autenticación adaptables vía JAAS
- Caché: utiliza cacheado en la capa de visualización para mejor rendimiento
- Clusterizable: soporte de Cluster que permite que un portal pueda ser desplegado en varias instancias
- Hot-Deployment: hace uso de las características de autodespliegue dinámico incluido en JBoss
- Instalador SAR: instalación basada en web que hace que la instalación y configuración inicial sea muy sencilla.

Estándares soportados

- Portlet Specification and API 1.0 (JSR-168)
- Content Repository for Java Technology API (JSR-170)
- Java Server Faces 1.2 (JSR-252)
- Java Management Extensión (JMX) 1.2
- Compatibilidad 100% con J2EE 1.4 al utilizar JBoss AS

Contenedor de Portales

- Múltiples Instancias de Portales: habilidad para ejecutar múltiples portales desplegados en un único contenedor.
- IPC (Inter-Portlet Communication): la API habilita a los portlets crear enlaces a otros objetos como páginas, portales o ventanas.
- Dynamicity: permite a administradores y usuarios crear y eliminar objetos como portlets, páginas, portales, temas y composición en tiempo de ejecución.
- Internacionalización: permite utilizar recursos de internacionalización para cada portlet.
- Servicios empotrables: la autenticación realizada por el contenedor de servlets y JAAS posibilita cambiar el esquema de autenticación.
- Arquitectura basada en Páginas: permite for the grouping/division of portlets on a per-page basis.
- Soporte de Frameworks existentes: los Portlets pueden utilizar Struts, Spring MVC, Sun JSF-RI, AJAX o MyFaces.

Temas y Layouts

- Temas y Layouts fácilmente intercambiables: los temas y layouts nuevos que contienen imágenes se pueden desplegar en ficheros WAR.
- API Flexible: la API de Temas y Layout están diseñados para separar la lógica de negocio de la capa de presentación.
- Estrategia de layout por página: a cada página se le puede asignar layouts distintos.

Funcionalidades de Usuarios y Grupos

- Registro y validación de usuarios: parámetros configurables del registro permite la validación de usuarios vía email previa a la activación.
- Acceso de usuarios: hace uso de la autenticación del contenedor de servlets.
- Crear/Modificar usuarios: habilita a los administradores crear/modificar perfiles de usuarios.
- Crear/Modificar roles: habilita a los administradores crear/modificar roles.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

- Asignación de roles: habilita a los administradores asignar roles a los usuarios.

Gestión de Permisos

- API extensible de permisos: permite asignar permisos de acceso a portlets basados en la definición de roles.

- Interfaz de administración: asignación de permisos a roles en cualquier momento para portlets, páginas o instancias de portal desplegados.

Sistema de gestión de contenidos

- Compatible JCR: el CMS utiliza Apache Jackrabbit, una implementación en código abierto del estándar Java Content Repository API.
- Soporte de almacenamiento en SGBD o en el sistema de ficheros.
- Soporte externo de contenidos tipo Blob (binarios): se puede configurar el almacenamiento en el sistema de ficheros de contenido binario de gran tamaño y los nodos con las referencias y propiedades residan en el SGBD.
- Control de versiones: Todo contenido modificado/creado es autoversionado con el historial de cambios, que pueden ser revisados en cualquier momento.
- Contenidos mostrados en URLs amigables para los motores de búsqueda: http://yourdomain/portal/content/index.html (sin incluir las acciones de los portlets)
- URLs del portal sencillas: mostrar descarga de binarios con URLs fáciles de recordar. (http://domain/files/products.pdf)
- Soporte de múltiples instancias de Portlets HTML: permite que instancias extra de contenido estático del CMS sean publicadas en ventanas distintas.
- Soporte de directorios: crear, mover, eliminar, copiar y subir árboles completos de directorios.
- Funciones de Ficheros: crear, mover, copiar, cargar y eliminar ficheros.
- Explorador de directorios embebido: cuando se copia, mueve, elimina o se crean nuevos ficheros, los administradores pueden simplemente navegar por el árbol de directorios hasta encontrar la colección para que quieren realizar la acción.
- Arquitectura fácil de usar: todas las acciones que se pueden realizar sobre ficheros pueden hacerse a base de clicks de ratón.
- Editor HTML: con modo WYSIWYG, previsualización y edición de código HTML. Soporta la creación de tablas, fuentes, zoom, enlaces a imágenes y URLs, soporte de películas flash, listas con viñetas o numéricas...
- Soporte de editor de estilos: el editor WYSIWYG muestra la hoja de estilo actual del Portal, para un sencillo intercambio de clases.
- Soporte de Internacionalización: los contenidos pueden ser asignados para una zona regional determinada y ser mostrada en función de la configuración de usuario o basado en las opciones del navegador web .

Tablón de mensajes

- Respuesta inmediate mediante un sólo click.
- Respuesta con cita: se puede citar un tema existente al responder.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

- Control del flujo: previene el abuso de envío masivo de mensajes mediante una ventana de tiempo configurable.
- Creación de categorías contenedoras de foros.
- Operaciones sobre Foros: se puede crear un foro y asignarlo a una categoría específica, además se puede copiar, mover, modificar y eliminar.
- Reordenación de foros y categorías: se puede establecer el orden en el que se quiere que aparezcan los foros y categorías en las páginas.

Otras alternativas

Dentro del mundo Java existen otras alternativas opensource para la creación de portales que contienen todos los servicios anteriormente mencionados por las otras plataformas y que cumplen los estándares de industria, y que son alternativas perfectamente válidas para un proyecto de grandes prestaciones.

- Open nuke: https://opennuke.dev.java.net/
- LifeRay: http://www.liferay.com/web/guest/home
- InfoGlue: http://www.infoglue.org/infoglueDeliverLive/
- Apache Lenya: http://lenya.apache.org/index.html
- Apache JetSpeed: http://portals.apache.org/

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Páginas web de consulta.

- Área de descargas de BEA
- [http://commerce.bea.com/index.jsp]
- Área de descargas de ATG
- [http://www.atg.com/en/myatg/mydownloads.jhtml]
- The Server Side: Web de recursos sobre servidores de aplicaciones basados en J2EE.
- [http://theserverside.com]

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

References

1. Adrián Sánchez-Carmona, Sergi Robles, Carlos Borrego (2015). Improving Podcast Distribution on Gwanda using PrivHab: a Multiagent Secure Georouting Protocol. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
2. Anderson Sergio, Sidartha Carvalho, Marco Rego (2014). On the Use of Compact Approaches in Evolution Strategies. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
3. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. Information Sciences, 180(10), 2029–2043. https://doi.org/10.1016/j.ins.2009.12.032
4. Buciarelli, E., Silvestri, M., & González, S. R. (2016). Decision Economics, In Commemoration of the Birth Centennial of Herbert A. Simon 1916-2016 (Nobel Prize in Economics 1978): Distributed Computing and Artificial Intelligence, 13th International Conference. Advances in Intelligent Systems and Computing (Vol. 475). Springer.
5. Canizes, B., Pinto, T., Soares, J., Vale, Z., Chamoso, P., & Santos, D. (2017). Smart City: A GECAD-BISITE Energy Management Case Study. In 15th International Conference on Practical Applications of Agents and Multi-Agent Systems PAAMS 2017, Trends in Cyber-Physical Multi-Agent Systems (Vol. 2, pp. 92–100). https://doi.org/10.1007/978-3-319-61578-3_9
6. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
7. Casado-Vara, R., de la Prieta, F., Prieto, J., & Corchado, J. M. (2018, November). Blockchain framework for IoT data quality via edge computing. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (pp. 19-24). ACM.
8. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
9. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
10. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
11. Chamoso, P., de La Prieta, F., Eibenstein, A., Santos-Santos, D., Tizio, A., & Vittorini, P. (2017). A device supporting the self-management of tinnitus. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 10209 LNCS, pp. 399–410). https://doi.org/10.1007/978-3-319-56154-7_36
12. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
13. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
14. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
15. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
16. Choon, Y. W., Mohamad, M. S., Deris, S., Illias, R. M., Chong, C. K., Chai, L. E., … Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. PLoS ONE, 9(7). https://doi.org/10.1371/journal.pone.0102744
17. Constantino Martins, Ana Rita Silva, Carlos Martins, Goreti Marreiros (2014). Supporting Informed Decision Making in Prevention of Prostate Cancer. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
18. Costa, Â., Novais, P., Corchado, J. M., & Neves, J. (2012). Increased performance and better patient attendance in an hospital with the use of smart agendas. Logic Journal of the IGPL, 20(4), 689–698. https://doi.org/10.1093/jigpal/jzr021

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

19. David Griol, Jose Manuel Molina, Araceli Sanchís De Miguel (2014). Developing multimodal conversational agents for an enhanced e-learning experience. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
20. Eva L. Iglesias, Lourdes Borrajo, R. Romero (2014). A HMM text classification model with learning capacity. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
21. Fernández-Riverola, F., Díaz, F., & Corchado, J. M. (2007). Reducing the memory size of a Fuzzy case-based reasoning system applying rough set techniques. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 37(1), 138–146. https://doi.org/10.1109/TSMCC.2006.876058
22. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. Expert Systems with Applications, 41(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003
23. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. BMC Bioinformatics, 10. https://doi.org/10.1186/1471-2105-10-187
24. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
25. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
26. González-Briones, A., De La Prieta, F., Mohamad, M., Omatu, S., & Corchado, J. (2018). Multi-agent systems applications in energy optimization problems: A state-of-the-art review. Energies, 11(8), 1928.
27. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
28. Hafewa Bargaoui, Olfa Belkahla Driss (2014). Multi-Agent Model based on Tabu Search for the Permutation Flow Shop Scheduling Problem. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
29. Jamal Ahmad Dargham, Ali Chekima, Ervin Gubin Moung, Sigeru Omatu (2014). The Effect of Training Data Selection on Face Recognition in Surveillance Application. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
30. Jorge Agüero, Miguel Rebollo, Carlos Carrascosa, Vicente Julián (2013). MDD-Approach for developing Pervasive Systems based on Service-Oriented Multi-Agent Systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
31. Juan Castro, Pere Marti-Puig (2014). Real-time Identification of Respiratory Movements through a Microphone. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
32. Leonor Becerra-Bonache, M. Dolores Jiménez López (2014). Linguistic Models at the Crossroads of Agents, Learning and Formal Languages. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
33. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
34. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). Random finite set-based Bayesian filters using magnitude-adaptive target birth intensity. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637788&partnerID=40&md5=bd8602d6146b014266cf07dc35a681e0
35. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. Applied Mathematics and Computation, 270, 756–767. https://doi.org/10.1016/j.amc.2015.08.059
36. Margherita Brondino, Gabriella Dodero, Rosella Gennari, Alessandra Melonio, Daniela Raccanello, Santina Torello (2014). Achievement Emotions and Peer Acceptance Get Together in Game Design at School. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

37. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. Expert Systems with Applications, 36(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
38. Miki Ueno, Naoki Mori, Keinosuke Matsumoto (2014). Picture models for 2-scene comics creating system. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
39. Ming Fei Siyau, Tiancheng Li, Jonathan Loo (2014). A Novel Pilot Expansion Approach for MIMO Channel Estimation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
40. Omar Jassim, Moamin Mahmoud, Mohd Sharifuddin Ahmad (2014). Research Supervision Management Via A Multi-Agent Framework. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
41. Pablo Chamoso, Henar Pérez-Ramos, Ángel García-García (2014). ALTAIR: Supervised Methodology to Obtain Retinal Vessels Caliber. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
42. Paula Andrea Rodríguez Marín, Néstor Duque, Demetrio Ovalle (2015). Multi-agent system for Knowledge-based recommendation of Learning Objects. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
43. Rodríguez, S., De La Prieta, F., Tapia, D. I., & Corchado, J. M. (2010). Agents and computer vision for processing stereoscopic images. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6077 LNAI). https://doi.org/10.1007/978-3-642-13803-4_12
44. Rodríguez, S., Gil, O., De La Prieta, F., Zato, C., Corchado, J. M., Vega, P., & Francisco, M. (2010). People detection and stereoscopic analysis using MAS. In INES 2010 - 14th International Conference on Intelligent Engineering Systems, Proceedings. https://doi.org/10.1109/INES.2010.5483855
45. Román, J. A., Rodríguez, S., & de da Prieta, F. (2016). Improving the distribution of services in MAS. Communications in Computer and Information Science (Vol. 616). https://doi.org/10.1007/978-3-319-39387-2_4
46. Sigeru Omatu, Tatsuyuki Wada, Pablo Chamoso (2013). Odor Classification using Agent Technology. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
47. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. International Journal of Ambient Computing and Intelligence, v 1, n 1(1), 15–26. https://doi.org/10.4018/jaci.2009010102

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc