UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Proyecto final de carrera

Extracción de datos web usando técnicas de screen-scraping.

Autor: Javier Ibáñez Micó
jaibmi@ei.upv.es
Dirigido por: Joan Fons i Cors
jjfons@dsic.upv.es

Índice

1.- Introducción

Cada día la necesidad del acceso a la información aumenta considerablemente. El crecimiento y expansión de la conocida como "red de redes" nos ha llevado a la llamada era de la información. Todo aquello que necesitamos saber o conocer lo encontramos en Internet: información meteorológica, últimas noticias, cambios en el mercado bursátil, etc. Y si se trata de una manera sencilla, cómoda y rápida mucho mejor.

Puede ocurrir que en determinadas ocasiones, tan solo estemos interesados en parte del contenido de una página web, deseando ignorar el resto de la información. Por ejemplo, en una página de meteorología, únicamente queremos conocer el tiempo actual de la ciudad donde nos encontramos. Y a su vez, sacar algún provecho de los datos que recuperemos, es decir, podríamos utilizar los datos de la situación meteorológica actual para la gestión automática de las persianas de una vivienda inteligente. O bien, mostrarla en nuestro terminal móvil y, así, conocer la previsión del tiempo de una manera usable.

Este proceso de extracción de una porción de una web es conocido como "Screen-scraping". Según la Wikipedia, Screen-scraping es el nombre en inglés de una técnica de programación que consiste en tomar una presentación de una información (normalmente texto, aunque puede incluir información gráfica) para, mediante ingeniería inversa, extraer los datos que dieron lugar a esa presentación.

Web-scraping es una técnica de programación para la extracción de información de sitios web. Por lo general, este tipo de programas simulan la exploración humana de la Web, ya sea implementado a bajo nivel (HTTP), o incluido en ciertos navegadores web, como el Internet Explorer (IE) o el navegador web Mozilla. Web-scraping está estrechamente relacionado con la indexación web, indexa todo el contenido de la Web mediante un robot y es una técnica universal adoptada por la mayoría de los motores de búsqueda. Web-scraping se centra en la transformación de contenido web no estructurado, por lo general en formato HTML, en datos estructurados que pueden ser almacenados y analizados en una base de datos local o de hoja de cálculo. Web-scraping también está relacionado con la automatización web, que simula la navegación humana web utilizando el software de computadora. Ejemplos de usos de Web-scraping son: la comparación de precios en línea, seguimiento del tiempo de datos, detección de cambios sitio web, la búsqueda en Internet, la integración de datos Web, etc.

Se trata de un proceso de recogida automática de información de la Web, a priori pensada para ser mostrada y no utilizada por otros sistemas. Web-scarping es un campo con una evolución activa que comparte un objetivo común con la visión de web semántica, una ambiciosa iniciativa que aún requiere de avances en el procesamiento de texto, la comprensión semántica, inteligencia artificial y las interacciones humano-computadora. Web-scraping, en cambio, busca soluciones prácticas basadas en las tecnologías existentes. Por lo tanto, hay diferentes niveles de automatización que las actuales tecnologías de Web-scraping pueden proporcionar.

En este proyecto se ha utilizado Screen-scraping para obtener información de diferentes páginas web y juntar todos los datos recogidos en una misma web generada dinámicamente. Una de sus características principales radica en hecho de que si por motivos externos a la aplicación, como por ejemplo una caída del servidor en el que está almacenada la página sobre la cual queremos extraer la información, no podemos obtener parte de los datos que necesitamos, la aplicación funcionará correctamente mostrando el resto de los valores. Esta particularidad nos la facilitara OSGI, que nos proporcionará una pasarela que facilitará la comunicación entre los diferentes servicios que necesitemos crear.

A lo largo del presente documento trataré de explicar como ha sido el proceso de desarrollo software que he llevado a cabo, es decir, el análisis, diseño e implementación del sistema, así como una explicación del caso de estudio concreto que supone el motivo de esta memoria y las conclusiones obtenidas al final de todo este proceso de desarrollo de mi proyecto final de carrera. Para todo ello seguiré el siguiente guión:

- Introducción
- Contexto tecnológico
- Análisis del sistema
- Diseño del sistema
- Implementación
- Conclusiones

2.- Contexto tecnológico

En el apartado anterior, se ha comentado que el campo de Web-scraping esta en evolución, existen numerosas tecnologías que se podrían haber utilizado para la extracción de datos web. No obstante, por motivos de requisitos, diseño e implementación, se ha optado por las que se van a describir a continuación. Tratemos ahora de explicar brevemente cada una de las diferentes tecnologías que hemos utilizado para el desarrollo del proyecto.

2.1 OSGI:

La Open Service Gateway Initiative (OSGi) es una asociación de empresas creada con el objetivo de definir un estándar abierto para el desarrollo de pasarelas residenciales. Inicialmente fueron 15 las compañías que fundaron esta asociación, entre las que destacan: Sun Microsystems, IBM, Lucent Technologies, Motorola, Ericsson, Toshiba, Nortel Networks, Oracle, Philips, Sybase, Toshiba, entre otras. Ahora son más de 80 las empresas que pertenecen a esta asociación. Hay fabricantes de hardware o PCs, empresas de software, de sistemas de gestión corporativos, operadores de telecomunicaciones, hasta varias compañías eléctricas. Actualmente ya tiene socios españoles como Unión Fenosa y Telefónica I+D.

OSGi pretende ofrecer una arquitectura completa, de extremo-a-extremo, que cubra todas las necesidades del proveedor de servicios, del cliente y de cualquier dispositivo instalado en las viviendas. Define arquitectura software mínima necesaria para que todos los servicios se puedan ejecutar en la misma plataforma independientemente del hardware usado (microprocesador, memoria, periféricos, modems, etc.). De esta forma permite a cualquier fabricante decidir cómo y dónde instala este software en plataformas compatibles que sean capaces de proporcionar múltiples servicios en el ámbito de la vivienda.

Así pues, OSGI puede describirse como una colección de APIs basados en Java que permiten el desarrollo de servicios independientemente de la plataforma. Estas APIs permiten la compartición de los servicios, el manejo de datos, recursos y dispositivos, el acceso de clientes y la seguridad. Como la especificación OSGI está orientada a la capa de aplicación se puede complementar con cualquiera de los protocolos de comunicación y redes domóticas existentes, tales como Bluetooth, EIB, CEBus, Home API, HomePNA, HomePNP, HomeRF...

Describimos a continuación los aspectos más importantes de la especificación de OSGI.

2.1.1 Especificación

El núcleo de la especificación de OSGI es el Framework (Imagen 1) que proporciona un entorno estandarizado para las aplicaciones (conocidas como bundles). Se divide en 4 capas (Imagen 2):

Imagen 1: Framework de OSGI

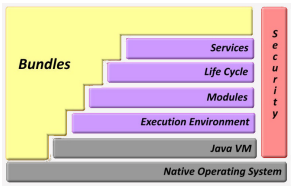


Imagen 2: Capas de OSGI

- Entorno de ejecución (L0): es la especificación del entorno de Java. La configuración de Java 2 y sus perfiles como J2SE, CDC, CLDC, MIDP, etc. Son válidos todos los entornos de ejecución. La plataforma OSGi también ha estandarizado un entorno de ejecución basado en un perfil y una pequeña variación que especifica los requisitos mínimos en un entorno de ejecución para ser útil frente a los bundles OSGI.

- Módulos (L1): define las políticas de dependencias en la clase de carga. Se basa en la parte superior de Java, pero añade modularización. En Java, normalmente existe un único classpath que contiene todas las clases y los recursos. L1 añade clases particulares para un módulo y se encarga de controlar la vinculación entre módulos. Además, está totalmente integrado con la arquitectura de seguridad, lo que permite la opción de desplegar sistemas cerrados o sistemas de gestión de usuario.

- Gestión del ciclo de vida (L2): esta capa permite que los bundles puedan ser instalados, iniciados, parados, modificados o desinstalados dinámicamente. Un

activador es la clase que se encarga de gestionar el ciclo de vida de un Bundle. Esta clase tiene unos métodos start y stop que son invocados cuando el Bundle se arranca y detiene, respectivamente.

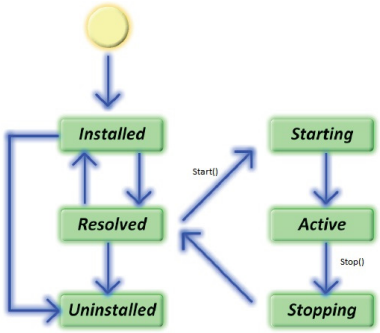


Imagen 3: Ciclo de vida de un bundle

Estado	Descripción
Installed	El bundle se ha instalado correctamente.
Resolved	Todas las dependencias del bundle están satisfechas y el bundle está preparado para ser iniciado (o ya ha sido iniciado y parado).
Starting	El bundle está siendo iniciado, es decir, se ha llamado al método start() pero este aun no ha terminado de ejecutarse.
Active	El bundle ha sido activado e iniciado correctamente.
Stopping	Se ha llamado al método stop() del bundle pero esta aún se está ejecutando.
Uninstalled	El bundle ha sido desinstalado y no puede pasar a ningún otro estado.

Imagen 4: Estados del ciclo de vida

El Framework OSGi en combinación con el contexto, manejan el llamado ciclo de vida de los bundles (Imagen 3) instalados en el sistema. Esto no es más que una sucesión de estados (Imagen 4) cuyas transiciones entre estos determinas las acciones que pueden ejecutarse.

- Registro de Servicios (L3): permite que los bundles puedan compartir objetos. Un servicio no es más que un objeto Java, registrado como proveedor de una determinada interfaz, que puede ser simplemente un objeto software o puede representar un objeto del mundo real. Al ser registrado un servicio, se le pueden asociar una serie de propiedades que podrán ser utilizadas por otros servicios para realizar búsquedas concretas. Algunas de estas propiedades son obligatorias, como el PID, que sirve para identificar a un servicio durante toda su actividad en OSGi. Además permite que se puedan definir acciones a realizar cuando se registra o se deja de ofrecer el servicio.

- Bundles (L4): Los bundle son el mecanismo utilizado para distribuir e instalar aplicaciones y servicios en OSGi. Un bundle es un archivo Java (JAR) con la parte mínima de una aplicación OSGi, además, tienen un ciclo de vida restringido que es administrado externamente por el framework. El registro de servicios provee el mecanismo de manejo de los mismos, por eso, un bundle está diseñado para ser un conjunto de servicios que cooperan en el registro de servicios.

2.1.2 ¿Cómo funciona OSGI?

- Manifest.mf: El archivo manifest, normalmente está ubicado en la carpeta META-INF y su nombre debe ser MANIFEST.MF. Este archivo contiene varias etiquetas entre las que cabe destacar las siguientes:
  - Bundle-Activator: ruta donde se encuentra la clase activator.
  - Bundle-Name: nombre del servicio.
  - Bundle-Description: descripción del bundle.
  - Import-Package: paquetes que necesita para ejecutarse, se requerirá la mayoría de las veces, que se importe *org.osgi.framework*.

- Activator: La clase Activator sirve para personalizar el arranque y la parada de un bundle. Debe cumplir con la interfaz BundleActivator y sobrescribir los métodos:

```
public void start(BundleContext context);
public void stop(BundleContext context);
```

  El método `start(BundleContext context)` se ejecutará cuando el framework inicie el bundle, y el método `stop (BundleContext context)` cuando lo detenga.

- Registro de servicios: En OSGi existe el concepto de "contexto" en el cual se mantiene la información de los servicios (interfaces, implementaciones, información, etc.). A este contexto se le puede:
  - Añadir nuevos servicios (registrar)
  - Consultar por servicios existentes (consultar)

  Cada componente de OSGI (bundle) recibirá el contexto en el que se está ejecutando a través del Activator del bundle. Este contexto es de tipo BundleContext, que se encuentra en el paquete: org.osgi.framework.BundleContext.
  En el contexto de OSGi se pueden buscar implementaciones (registradas previamente) de interfaces (servicios).
  Para buscar un servicio, se debe indicar:
  - el interfaz que se busca (clazz)
  - un filtro (opcional) en base a las propiedades registradas del servicio, usando notación LDAP (filter)

  OSGi proporciona dos operaciones de consulta:
  - Obtener (referencia a) un servicio del contexto

```
ServiceReference
context.getServiceReference(clazz, filter);
```

  - Obtener todos (referencias a) los servicios …

```
ServiceReference[]
context.getAllServiceReferences(clazz, filter);
```

- Comunicaciones OSGI: Existen varios modelos de comunicación en OSGi aunque con la misma finalidad, comunicar 2 o más servicios que en un momento dado necesitan compartir cierta información. De entre las más importantes cabe destacar los *Wire y* los *ServiceListener.*

  - Wire – Producer & Consumer.
    - *Producer:* Un servicio Producer es aquel que genera datos para que sean consumidos por un Consumer. Estos servicios deben ser implementados por una clase que cumpla con la interfaz Producer. Cuando un dato gestionado por el Producer cambie, debe avisar a todos los Consumer conectados mediante el método update().

```
public void consumersConnected(Wire[] wires);
public Object polled(Wire wire);
```

- *Consumer:* Los Consumer son servicios que pueden recibir valores actualizados por un Producer. Estos servicios deben ser implementados por una clase que cumpla con la interfaz Consumer y sobrescribir los métodos:

```
public void producersConected(Wire[] wires);
public void updated(Wire wire, Object value);
```

  Los servicios que se registren como Consumer, recibirán valores vía Wire desde uno de los Producer a que se haya asociado.

- *Wire:* Un objeto Wire sirve para conectar un servicio *Producer* con un servicio *Consumer*. Ambos son identificados con un PID y pueden comunicarse con el otro vía los objetos que el *Producer* introduce dentro del *wire* para que el *Consumer* los reciba. Cabe decir que 2 servicios pueden estar comunicados con múltiples wire e incluso estar conectados a terceros. Otro aspecto interesante es que un tercer servicio puede ser el que cree el wire entre otros dos servicios *Producer* y *Consumer*.

- ServiceListener: Un ServiceListener, es una clase que implementa la misma interfaz y que se añade al BundleContext precisamente como Listener de eventos de todos los Servicios que cumplan cierta interfaz. Los tipos de eventos que se detectaran serán:

```
public void serviceChanged(ServiceEvent event);
public class ServiceEvent{
public static int MODIFIED;
public static int MODIFIED_ENDMATCH;
public static int REGISTERED;
public static int UNREGISTERED;
```

2.2 XQuery:

Extensible Markup Language (más conocido como XML) es un formato para almacenar datos extremadamente versátil, utilizado para representar una gran cantidad distinta de información como, por ejemplo, páginas web (XHTML).

Un conjunto de datos expresados en XML es una cadena de texto donde cada uno de los datos está delimitado por etiquetas de la forma <T>…</T> o se incluye como atributo de una etiqueta de la forma <T A="...">...</T>.

Actualmente, XML se ha convertido en una herramienta de uso cotidiano en los entornos de tratamiento de información y en los entornos de programación.

El escenario de uso más común se da cuando tenemos que realizar alguna búsqueda en un documento o conjunto de documentos con gran cantidad de datos en XML, o bien hemos de trabajar sólo sobre un subconjunto de todos los datos XML y queremos obtener dicho subconjunto de una forma sencilla, para evitar trabajar con toda la colección de datos.

Dar solución a este escenario de uso mediante código escrito con la ayuda de un parser SAX es rápido y sencillo de desarrollar si la consulta y la estructura de la información no tienen una gran complejidad. A medida que la consulta va ganando en complejidad, es necesario definir comportamientos más complejos a ejecutar cuando se encuentre una etiqueta y se necesita un mayor número de almacenamientos temporales de datos. Otro problema es que el código está fuertemente ligado a la estructura de los datos en XML, por lo que cualquier cambio en la consulta o en la estructura de los datos obliga a volver a escribir el código. Además el código resultante es muy poco parametrizable y reutilizable por lo que para desarrollar una consulta similar habría que volverla a escribir desde el principio, sin poder aprovechar nada de lo escrito.

De manera rápida podemos definir XQuery con un símil en el que XQuery es a XML lo mismo que SQL es a las bases de datos relacionales. XQuery es un lenguaje de consulta diseñado para escribir consultas sobre colecciones de datos expresadas en XML. Abarca desde archivos XML hasta bases de datos relacionales con funciones de conversión de registros a XML. Su principal función es extraer información de un conjunto de datos organizados como un árbol n-ário de etiquetas XML. En este sentido XQuery es independiente del origen de los datos.

XQuery es un lenguaje funcional, lo que significa que, en vez de ejecutar una lista de comandos como un lenguaje procedimental clásico, cada consulta es una expresión que es evaluada y devuelve un resultado, al igual que en SQL. Diversas expresiones pueden combinarse de una manera muy flexible con otras expresiones para crear nuevas expresiones más complejas y de mayor potencia semántica.

XQuery está llamado a ser el futuro estándar de consultas sobre documentos XML. Actualmente, XQuery es un conjunto de borradores en el que trabaja el grupo W3C. Sin embargo, a pesar de no tener una redacción definitiva ya existen o están en proceso numerosas implementaciones de motores y herramientas que lo soportan.

Aunque XQuery y SQL puedan considerarse similares en casi la totalidad de sus aspectos, el modelo de datos sobre el que se sustenta XQuery es muy distinto del modelo de datos relacional sobre el que sustenta SQL, ya que XML incluye conceptos como jerarquía y orden de los datos que no están presentes en el modelo relacional. Por ejemplo, a diferencia de SQL, en XQuery el orden es que se encuentren los datos es importante y determinante, ya que no es lo mismo buscar una etiqueta <B> dentro de una etiqueta <A> que todas las etiquetas <B> del documento (que pueden estar anidadas dentro de una etiqueta <A> o fuera).

XQuery ha sido construido sobre la base de Xpath. Xpath es un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles XML. XQuery se basa en este lenguaje para realizar la selección de información y la iteración a través del conjunto de datos.

2.3 SAXON HE:

Saxon-HE (Home Edition) es un procesador de XSLT y XQuery creado por Michael Kay. Es un producto de código abierto disponible bajo la Licencia Pública de Mozilla. Proporciona implementaciones de XSLT 2.0, XQuery 1.0 y XPath 2.0 en el nivel básico de conformidad definidos por el W3C. Está disponible para Java y. NET.

Saxon permite convertir documentos XML en otros documentos XML; puede convertir un documento XML que obedezca a un DTD (Definición de Tipo de Documento) a otro que obedezca otro diferente, un documento XML bien formado a otro que siga un DTD.

Para este proyecto, nos será de gran interés convertir las páginas HTML, de las cuales queremos extraer la información, en XML-DOM. Para ello utilizaremos JTidy, con esta transformación simple, podremos procesar casi cualquier página Web como un documento XML, y posteriormente aplicar cualquier herramienta de extracción de datos.

2.4 Servlet:

Entre las muchas particularidades que distinguen a la Web de los restantes medios de comunicación, está la capacidad de interacción. Interacción que inicialmente era bastante reducida y se conseguía a través de código HTML y algunos elementos embebidos de JavaScript. Cuando se mencionaba el nombre de Java en este entorno, algunos lo asociaban a JavaScript, y otros, un poco más conocedores, lo asociaban con pequeños programas que se ejecutaban en el cliente y se llamaban Applets. Esto realmente fue el inicio de Java en la Web y surgió para darle más vida y dinamismo a la misma.

Esto resolvía casi todos los problemas al comienzo de la vida en la Web, y hacia el trabajo con Java bastante limitado pero, ¿realmente una página tiene que limitarse solo a contener textos y gráficos bonitos? Esta necesidad se comprendió muy pronto gracias al dinamismo de la informática y las exigencias de los usuarios, y comenzaron a aparecer tecnologías que convertían al usuario en la pieza principal de la aplicación, tecnologías que podían interpretar sus acciones y procesar datos no solo en el cliente si no también en el servidor. Los CGI de perl eran la solución en ese entonces. La comunidad de Java no podía quedarse atrás y se lanza la especificación original de Sevlet en junio de 1997, gracias al ingenio de los programadores de la SUN, en su versión 1.0.

Los Servlet son la respuesta de la tecnología en Java a la programación de la Interfaz de Compuerta Común (CGI). Son programas que se ejecutan en el servidor, realizando la función de una capa intermedia entre una petición proveniente de un navegador Web u otro cliente HTTP, y las aplicaciones del servidor, pudiendo utilizar toda la paquetería y potencialidades del lenguaje. Su función principal es proveer páginas web dinámicas y personalizadas, utilizando para este objetivo accesos a bases de datos, flujos de trabajo y otros recursos.

Para poder utilizar los Servlets dentro de nuestras aplicaciones web, debemos, por norma general, complementar éste con un contenedor de Servlets, que no es más que un servidor que tenga soporte para el API Servlet. En Internet disponemos de multitud de contenedores de código libre y comerciales como el Tomcat de Apache, el WebLogic de IBM, el JSWDK de la SUN, etc.

El API Servlet nos propone una jerarquía de clases (Imagen 5) bien definidas para el trabajo con los mismos, donde podemos encontrar desde servlets sin implementaciones definidas, en los cuales tenemos que realizar todo el trabajo como la clase GenericServlet, o clases un poco mas avanzadas como la HttpServlet, donde solo

tendríamos que implementar los métodos de acceso al mismo como el doGet() o el doPost(). El principal componente del API es la interfaz Servlet. La misma esta provista de los principales métodos para manipular, no solo los servlets, si no también la comunicación de estos con los clientes. Todos los servlets implementan esta interfaz de una manera directa o indirecta. A través de cualquiera de estas clases podemos comenzar a realizar nuestros propios servlets.

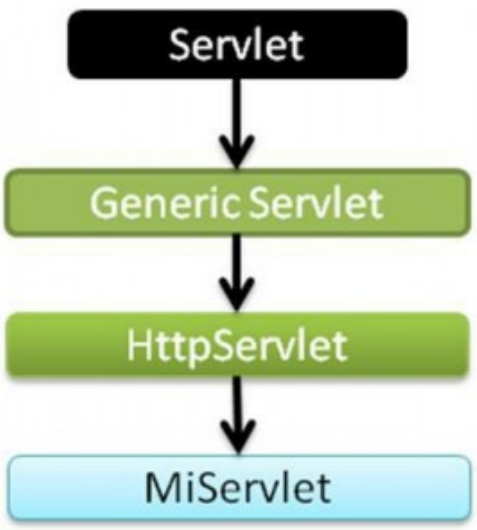


Imagen 5: Jerarquía de Servlet

Ventajas sobre los CGI tradicionales

En el momento que salió la especificación de Servet a la luz, los CGI ya llevaban algún tiempo e la red, lo que dio una ventaja muy grande pues permitió que fueran analizados a profundidad y de esta forma poder crear métodos con mejor rendimiento, potencialidad y facilidad de uso.

- Eficientes: Con el CGI tradicional, se arranca un nuevo proceso por cada petición HTTP. Si el propio programa CGI es relativamente pequeño, el costo de arrancar el proceso puede superar el tiempo de ejecución. Además, si ocurren n peticiones simultáneas al mismo programa este carga su código en memoria la misma cantidad de veces. Con los Servlet la maquina virtual de java permanece en ejecución y administra cada petición mediante ligeros subprocesos de Java y no un pesado proceso del sistema operativo. A través de la programación por hilos pueden existir n subprocesos del mismo Servlet y este encontrarse en memoria una sola vez. Esto nos da una velocidad de respuesta mayor a cada petición y una eficiencia superior en el aprovechamiento de los recursos de la maquina donde son ejecutados.

- Adecuados: Leer los datos enviados por un cliente HTTP desde un CGI tradicional es un trabajo bastante engorros, se necesita parsear la cadena completa y extraer de ella los datos uno a uno. Los Servlet cuentan con una extensa infraestructura para analizar y decodificar automáticamente los datos provenientes de un formulario HTML, leer y establecer encabezados HTTP, administrar las cookies, rastrear las sesiones y muchas otras utilidades de

alto nivel como estas. Además, si ya tiene algún dominio del lenguaje Java, se puede aprovechar sus conocimientos en este campo.

- Transportables: Los Servlet no son mas que una clase Java, por lo que pueden ser tan portables como cualquier aplicación escrita en el mismo lenguaje. Esto los convierte en multiplataforma por defecto por lo que pueden ser ejecutados en cualquier sistema operativo. Además cuentan con un API estándar. La API Servlet no incluye nada acerca de cómo son cargados los servlets, ni el ambiente en el cual corren los servlets, ni el protocolo usado para transmitir los datos del usuario. Esto permite a los servlets poder ser usados por diferentes servidores Web. Se pueden cargar indiferentemente y de forma transparente tanto desde un disco local como desde una dirección remota, de forma totalmente transparente. Es posible utilizarlos en servidores tan populares como el Apache, el FastTrack ó el Internet Information Server.
  Los Servlets pueden comunicarse entre sí, y por tanto, es posible una reasignación dinámica de la carga de proceso entre diversas máquinas. Es decir, un servlet podría pasarle trabajo a otro servlet residente en otra máquina conociendo solamente la URL de este.

- Seguros: Una de las principales fuentes de vulnerabilidad en los programas CGI tradicionales proviene del hecho de que por lo general son ejecutados por entornos de sistemas operativos de propósito general. Por ello, el programador de CGI debe tener cuidado de filtrar caracteres como las comillas tipográficas y los puntos y comas pues tienen un tratamiento especial por el entorno. Esto es más complejo de lo que podría pensarse, y los problemas derivados de esta situación son constantemente ignorados en diversas bibliotecas de CGI. Una segunda fuente de problemas es el hecho de que ciertos programas de CGI son procesados por lenguajes que no verifican automáticamente los límites de las matrices o cadenas. Por ejemplo, en C y C++ es perfectamente legal asignar una matriz de 100 elementos y describir en el "elemento" número 999, el cual puede ser un lugar aleatorio de la memoria del programa. Por ello, los programadores que olvidan realizar esta verificación, abren sus propios sistemas a posibles ataques de desbordamiento en el búfer ya sea por accidente o de manera deliberada. Los servlets no tienen estos problemas. Aun si un servlet ejecuta una llamada a un sistema distante para ejecutar un programa en el sistema operativo local, no utiliza el entorno del sistema operativo para lograrlo. Y por supuesto que la verificación de los límites de las matrices y otras características para la protección de la memoria es una parte central del lenguaje de programación Java.

El ciclo de vida de un Servlet

El proceso de ejecución de un servlet (Imágenes 6 y 7) comienza con la petición HTTP. Este llega en primer lugar al servidor web, el cual la traslada al motor del servicio Servlet/JSP. El motor encapsula la petición en un objeto del tipo HttpServletRequest y el flujo de respuesta en un objeto HttpServletRespopnse.

La interfaz ServletRequest permite al servlet acceder a la información pasada por el cliente como, los nombres de parámetros, el protocolo, los nombres de los host remotos que hacen la solicitud y el servidor que la recibe. Esta interfaz permite a los servlets el acceso a métodos que permiten manejar la presentación de la respuesta como salida en el navegador, a través de los cuales consiguen los datos desde el cliente que usa protocolos como HTTP POST.

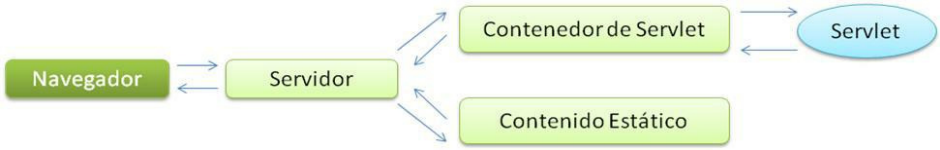


Imagen 6: Peticiones Servlet

La interfaz ServletResponse proporciona al servlet los métodos para responderle al cliente. Permite configurar el formato de salida de los datos. ServletOutputStream permite enviar la replica de datos como respuesta. Las subclases de ServletResponse le dan más capacidad al servlet para responder.

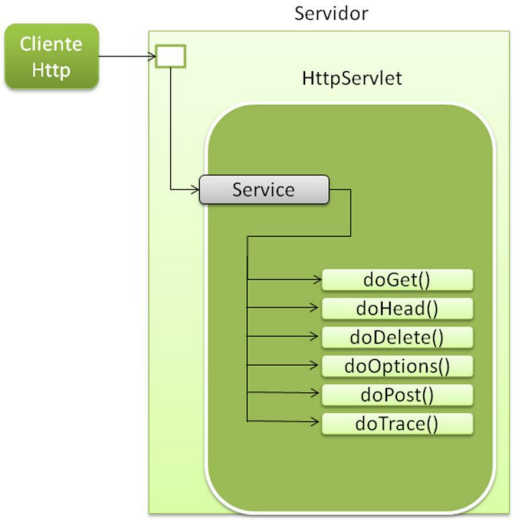


Imagen 7: Peticiones Servlet

El servlet es creado la primera vez que se invoca. Esto se hace a través del método **init()** que es el encargado de cargarlo en la memoria y pasarle los parámetros iniciales que se le hallan definido. Luego de que el servlet es inicializado, todas las demás peticiones serán tratadas en subprocesos o hilos que solo necesitaran como entrada el método **service()**. Este método invoca al doGet(), al doPost() o al método definido por el cliente para manejar la petición entrante.

Las peticiones serán manejadas como se ha explicado hasta ahora, pero ¿qué se hace con un servlet que no necesitamos que se ejecute más? ¿Lo tendríamos ocupando espacio en la memoria? Para resolver esta problemática existe el método **destroy()**. Este método es el encargado de destruir el servlet. Este es el momento indicado para cerrar las conexiones que existan con bases de datos u otros recursos compartidos, salvar el estado o lo que queramos que persista.

Las clases e interfaces descritas conforman a un servlet básico. Pero existen métodos adicionales que provee la API con la capacidad para controlar sesiones o múltiples conexiones, entre muchas más aplicaciones.

3.- Análisis del sistema

Como comentábamos en la introducción de este documento, el proyecto final de carrera motivo de esta memoria "Extracción de datos usando técnicas de Screen-Scraping", pretende conseguir que la recopilación y presentación de datos concretos, extraídos de diferentes web, se realice de una manera automática y limpia al usuario, así como asegurar que la información está totalmente actualizada.

Puesto que el propósito de este proyecto es el estudio de la extracción de datos mediante Screen-Scraping se han escogido los valores que se van a describir a continuación como ejemplo para demostrar que estas técnicas pueden aplicarse a cualquier campo cuya información esté disponible en la red. Del mismo modo que se han extraído estos mensajes, podríamos haber elegido diferentes valores de otras páginas web. A lo largo de esta memoria se hará referencia en diversas ocasiones a estos ejemplos de recolecta de datos, y serán tratados en mayor profundidad en el apartado "Caso de estudio".

3.1 Objetivos y requisitos de la aplicación.

3.1.1 Objetivo básico de la aplicación.

En concreto, la aplicación que se ha desarrollado, recopila la información referente a los valores meteorológicos (temperatura, velocidad del viento y humedad, así como la fecha y la hora de la última actualización de la página web de la cual extraemos los datos) de la ciudad de Ontinyent, el valor del IBEX 35 y una recopilación de los últimos titulares de la prensa.

En el caso de, por problemas externos a la aplicación, no encontrarse disponible parte del conjunto de datos a visualizar, el resto de información debería seguir mostrándose de una forma correcta y sin ninguna alteración. Es decir, cada bloque de datos debe ser tratado de manera independiente y no verse afectado por los demás.

Aunque posteriormente se va a explicar como aprovechar la aplicación desarrollada para ampliarla pudiendo de esta manera mostrar una mayor variedad y cantidad de datos, se puede ver claramente que tan solo vamos a ofrecer una funcionalidad al usuario de la aplicación. Mostrar mensajes de la captura de datos.

3.1.2 Especificaciones requisitos y funcionalidades.

- La aplicación deberá mostrar diferentes datos recopilados de varias páginas web, utilizando para ello técnicas de Screen-Scraping.
- La aplicación deberá mostrar la recopilación de datos en una página creada dinámicamente en tiempo de ejecución.
- Cada bloque de datos debe ser independiente del resto, esto implica que si un bloque no está disponible, por cualquier motivo externo a la aplicación, el resto seguirán funcionando correctamente.
- La arquitectura de la aplicación debe ser estable y garantizar la reusabilidad del código para poder ser ampliada.

3.2 Diagrama de clases.

Haremos uso de los diagramas de clases, un tipo de diagrama estático de UML que nos permitirá describir y visualizar las clases que componen el sistema y las relaciones que existen entre ellas. Se define una clase como una unidad básica que encapsula la información de un objeto, y se muestran atributos y métodos de las mismas.

Como en este momento no hemos llegado todavía a la fase de análisis, y con la intención de facilitar la comprensión en esta fase de análisis del sistema, describiremos a continuación una primera versión del diagrama de clases de esta aplicación.

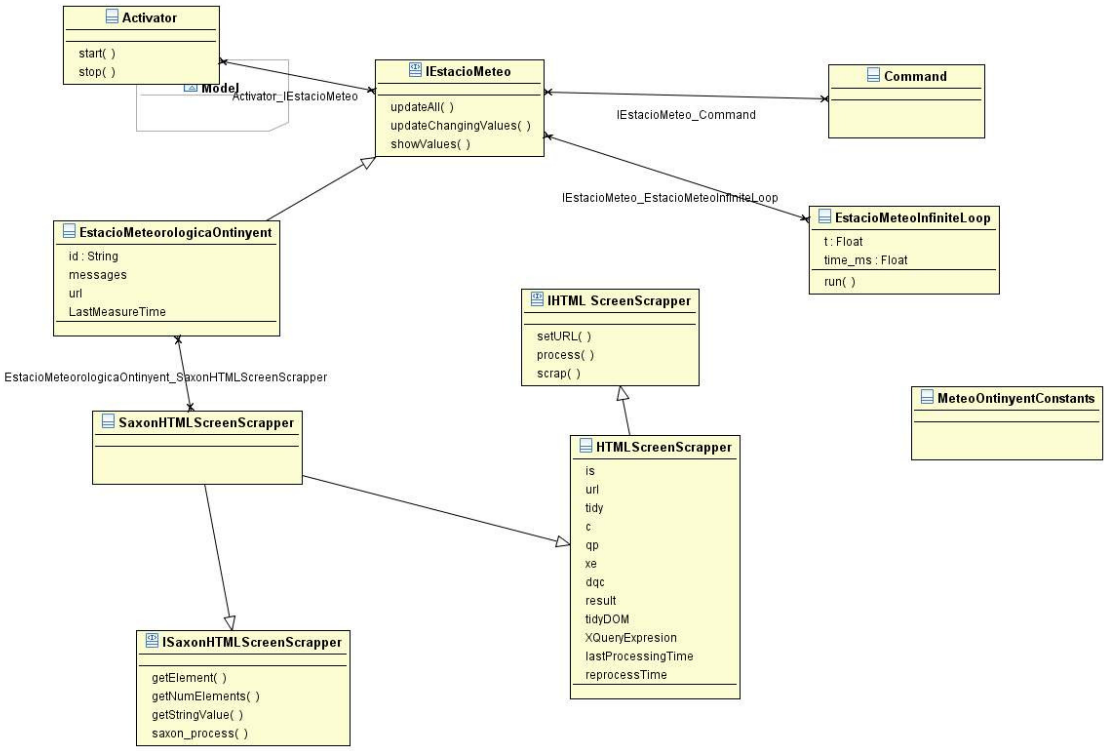


Imagen 8: Primera versión del diagrama de clases.

En la Imagen 8 se muestra la primera versión del diagrama de clases, como se observa, solamente hemos introducido una extracción de datos. Se trata de un intento de plasmar una primera aproximación de la envergadura de este proyecto y de facilitar el análisis para posteriormente poder realizar una mejor labor de diseño.

En concreto, este diagrama de clases expresa únicamente la parte de funcionalidad que respecta a la predicción meteorológica. En él contamos un total de 7 clases y 3 interfaces.

Las clases *SaxonHTMLScreenScrapper* y *HTMLScreenScrapper* son necesarias para todo el proceso de Scraping. En ellas se tratará el código HTML como XML para generar un DOM (Document Object Model) bien formado y ejecutar consultas XQuery sobre dicho documento, devolviéndonos el resultado de la misma como una cadena de texto.

Mientras que las clases *EstacioMeteorologicaOntinyent*, *EstacioMeteoInfiniteLoop* y *OntinyentMeteoConstants* son las encargadas de almacenar, solicitar y generar los mensajes con los resultados de las consultas XQuery. Así como de ir actualizando los valores de dichos mensajes regularmente y almacenar valores constantes (por ejemplo la dirección URL de la página web o las XQuery necesarias para obtener los datos).

Por último, la clase *Activator* es necesaria, como ya hemos comentado en el apartado “Contexto tecnológico”, para el correcto funcionamiento del bundle de OSGI. Y la clase *Command* se utiliza para poder establecer comunicación con la consola de Eclipse.

Es evidente pensar que si se continúa con esta política de crear un bundle por cada paquete de datos de una determinada página web, no estaremos reutilizando la mayor cantidad de código posible y aumentaríamos el tamaño del diagrama de clases considerablemente.

4.- Diseño del sistema

Vamos a ver a continuación las características del diseño de la aplicación. Para ello, presentaremos un diagrama de clases de la aplicación desarrollada en detalle, con todas las clases y relaciones definidas en este proyecto, como ampliación y refinación del presentado en el apartado de análisis. Realizaremos una descripción de las funcionalidades de cada clase. Veremos también en detalle las clases contenidas en cada bundle y un esquema de la interconexión entre cada uno de los bundles.

4.1 Aproximación a la solución final.

Tal y como se comentaba al final del apartado anterior, debemos replantear el problema y tratar de conseguir una mejor solución.

Si observamos con detalle el diagrama de clases anterior nos darnos cuenta de que se distinguen dos grandes bloques de clases, según la funcionalidad que implementan: infraestructura y aplicación (Imagen 9).

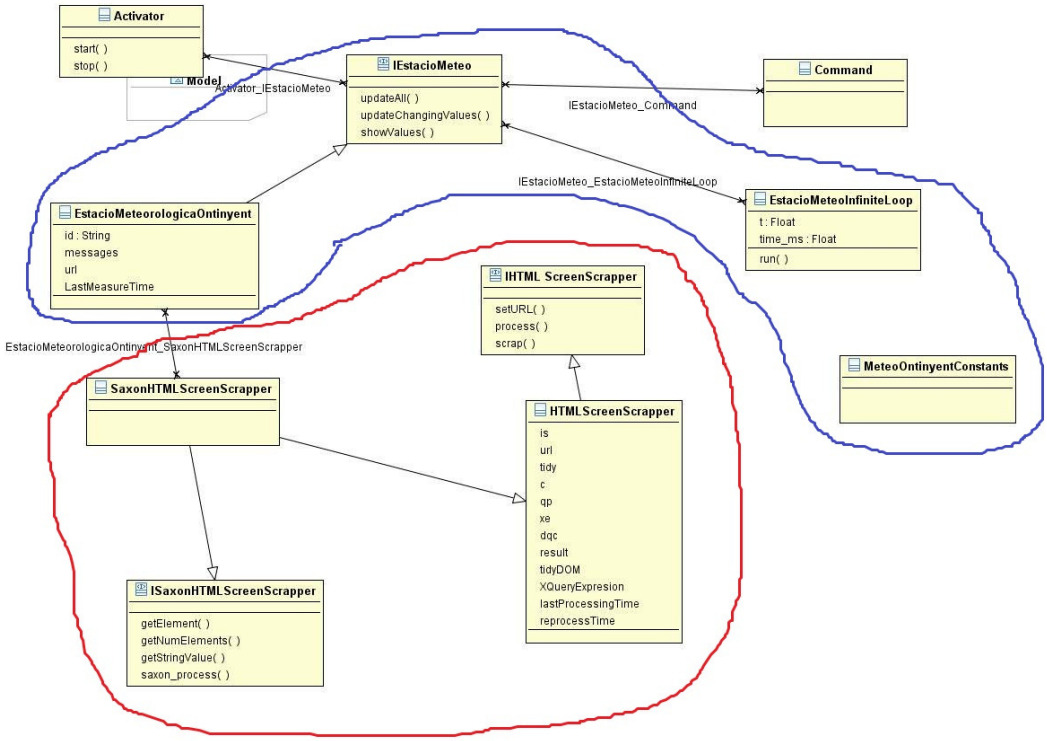


Imagen 9: Propuesta de división del diagrama.

Las entidades *EstacioMeteorologicaOntinyent, EstacioMeteoInfiniteLoop, MeteoOntinyentConstants* y *IEstacioMeteo* pertenecerían al bloque de aplicación, pues son entidades propias del conjunto de datos referente a la información meteorológica. Y por otra parte las entidades *SaxonHTMLScreenScrapper, HTMLScreenScrapper, ISaxonHTMLScreenScrapper* y *IHTMLScreenScrapper* formarían el bloque de entidades de infraestructura.

Razonando de este modo nos vamos a dar cuenta de que podemos aprovechar las características de OSGI para mejorar la propuesta de solución inicial. Planteemos ahora un caso en el que no se utilice únicamente un acceso a datos, es decir, contemplemos ahora que además de la información meteorológica deseemos también conocer el valor del IBEX 35 y un resumen de los titulares de última hora.

Nos damos cuenta de que tal vez la solución más eficiente seria utilizar un bundle para cada aplicación de acceso a datos, más otro de infraestructura y con otro más que gestionase los bundles de aplicación y se encargara también de generar la página web una vez recopilados los datos de los diferentes bundle-aplicación. Veamos un esquema para comprenderlo mejor.

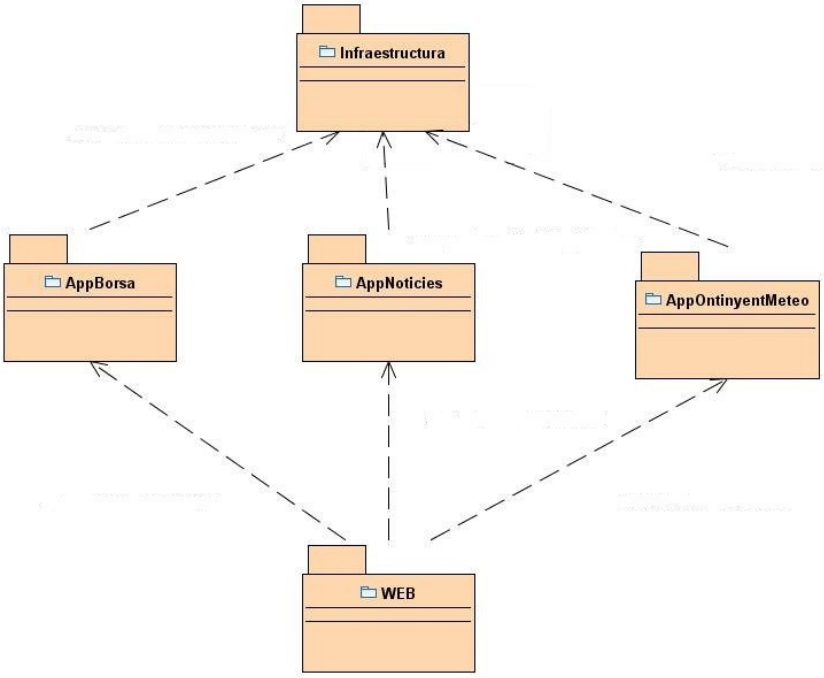


Imagen 10: Organización del caso de estudio por Bundles OSGI.

Lo que trata de representar el esquema de la Imagen 10, no es más que el conjunto de bundles de OSGI que se han implementado en el proyecto, así como la relación que existe entre ellos. De esta manera, queda claro que el bundle *WEB* solicitará a los bundles de aplicación los datos que hayan recopilado durante las ejecuciones de sus respectivas XQuery. Al mismo tiempo serán los bundles *AppBrosa*, *AppNoticies* y *AppOntinyentMeteo*, los que soliciten al bundle *Infraestructura* los servicios referentes y necesarios para poder ejecutar las XQuery que cada bundle de aplicación necesitan para completar su información.

Siguiendo este modo de diseño podremos asegurar que el código relativo a la tecnología Saxon, es decir, la reformación de la página web y la ejecución de las consultas, no se duplica en cada aplicación que deseemos crear, reaprovechando así el código en la medida de lo posible.

Por tanto, considerando esta aproximación que se acaba de exponer, podríamos tratar de crear un pseudo-diagrama de clases en el que no se pretende plasmar todo el significado que nos propone el modelo de UML, sino que su intención es dar una visión general del proyecto que se ha desarrollado. Es decir, no se van a tener en cuenta todos los tipos de datos de las diferentes propiedades, ni la noción de relación que propone el estándar, sino que simplemente se van a representar las clases e interfaces como clases, indicando qué métodos contiene cada una de ellas, así como sus propiedades y las relaciones con otras clases o interfaces.

Tomemos ahora cada uno de los paquetes que aparecen en la imagen 10 por separado y analicémoslo con más detenimiento comprendiendo su funcionamiento y contenido.

4.2 Análisis de los bundles.

Una vez explicada la relación entre cada uno de los diferentes bundles de OSGI, vamos a ver en detalle cual es su estructura interna y la relación existente entre sus diferentes partes.

Aclaremos que en la implantación del proyecto, las clases *Activator* de cada bundle reciben este mismo nombre. Por comodidad para el diseño del diagrama de clases se ha optado por asignar diferentes nombres a esta clase, es decir, las clases *ActivatorInfraestructura*, *ActivatorBorsa*, *ActivatorOntinyentMeteo*, *ActivatorNoticies* y *ActivatorWEB* en la implantación se nombran todas como *Activator*.

4.2.1 Infraestructura

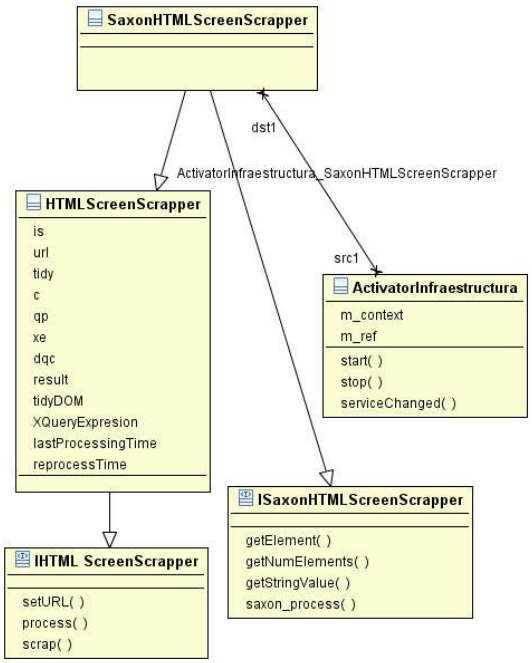


Imagen 11: Diagrama del contenido del bundle Infraestructura.

Como ya hemos comentado el bundle Infraestructura (Imagen 11) se encarga de tratar el código de las páginas web para generar un DOM bien formado sobre el que poder ejecutar las diferentes consultas XQuery.

Por un lado tenemos la clase *ActivatorInfraestructura* cuya función principal es mantener la operatividad del propio bundle.

Por otro lado, vemos la clase, de alguna manera, más importante de este paquete, que es *SaxonHTMLScreenScrapper*. Ella extiende a la clase *HTMLScreenScrapper* e implementa la interfaz *ISaxonHTMLScreenScrapper*. Su tarea es la de, gracias a las librerías de la tecnología Saxon, procesar la web para darle el formato DOM correcto y compilar y ejecutar las consultas sobre el código generado por ella misma.

4.2.2 AppBorsa, AppNoticies y AppOntinyentMeteo

Puesto que la similitud entre estos tres bundles es considerable, se va a explicar el funcionamiento del bundle AppBorsa (Imagen 12). Por lo que cualquier explicación sobre éste es totalmente extensible a los otros dos.

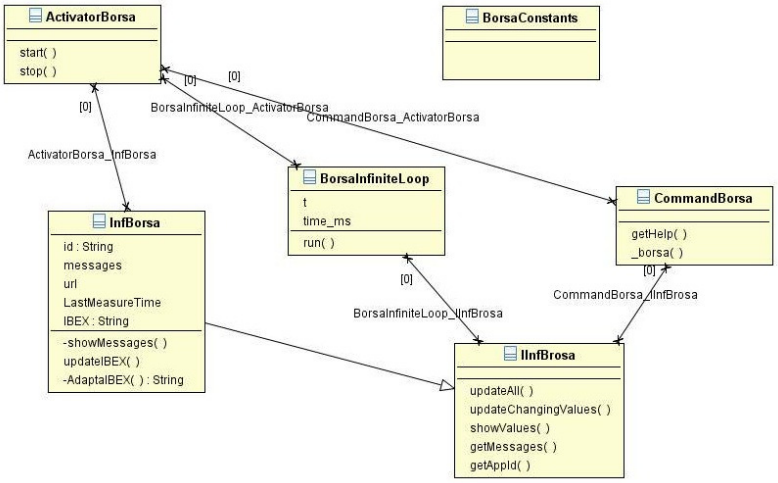


Imagen 12: Diagrama del contenido del bundle AppBorsa.

La principal tarea de este bundle consiste en solicitar al bundle infraestructura la ejecución de las XQuery que contiene en la clase *BorsaConstants* y almacenar el resultado en la lista de mensajes que serán después mostrados al usuario. La clase *InfBorsa* implementa la interfaz *IInfBorsa* y es la clase que contiene la mayor parte de la funcionalidad de esta aplicación. Por último la clase *BorsaInfiniteLoop* se encarga de volver a solicitar la ejecución de las XQuery, y por tanto de mantener actualizados los datos de la aplicación.

4.2.3 WEB

Por último, y no por ello menos importante, analicemos el contenido del bundle WEB (Imagen 13). Él será el encargado de coordinar las solicitudes de ejecución de las diferentes XQuery de las diferentes aplicaciones. Una vez hayan actuado dichas aplicaciones y se tenga toda la información, requerida por ellas, almacenada en los mensajes que se desean mostrar al usuario, se generará una página web dinámicamente (Servlet) en la que se muestre el contenido de toda la lista de mensajes.

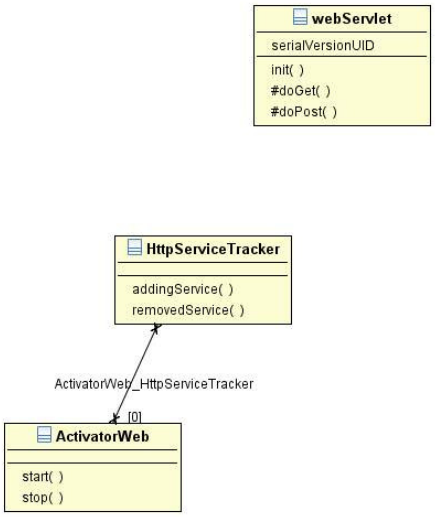


Imagen 13: Diagrama del contenido del bundle AppBorsa.

La clase *HttpServiceTracker* registra el Servlet en el espacio de nombre de un ORI. Y la clase *webServlet* contiene los métodos init(), doGet() y doPost(). Estos métodos se encargan de:

- init(): realizar acciones al realizarse la primera instancia al Servlet.
- doGet() y doPost(): se encargan de gestionar las peticiones de tipo GET y POST respectivamente.

Es la clase *webServlet* la encargada de coordinar las "llamadas" a los bundles de aplicación para que estos a su vez soliciten al bundle infraestructura la ejecución de las consultas.

La ejecución de este bundle esta íntimamente ligada con la instalación y configuración de un servidor de Servlet, en este caso se ha utilizado el servidor Tomcat en el que se ejecutará este bundle.

4.3 Ampliación del diagrama de clases.

Necesariamente, por cada aplicación que deseemos incorporar al proyecto, las dimensiones tanto del diagrama de clases, como del propio proyecto. Es por eso que se estudió la posibilidad de reducir el número de clases en los bundles aplicación para que de esta manera, el incremento del proyecto conforme al incremento de aplicaciones fuese menor.

Concretamente, la idea era crear un bundle nuevo que se encargara de contener básicamente la interfaz que se ha incorporado, finalmente, en cada bundle aplicación. Lamentablemente por problemas de comunicación entre los bundles no fue posible y se tuvo que optar por la solución que se propone a lo largo de este documento.

Ya se han analizado el contenido de los bundles que explicamos al inicio de este apartado por separado. Demos ahora un paso más allá y veamos una visión más global del proyecto implementado.

Una vez dicho esto, pasemos a ver el aspecto final del diagrama de clases y como interactúan entre sí las clases de los diferentes bundles en la Imagen 14.

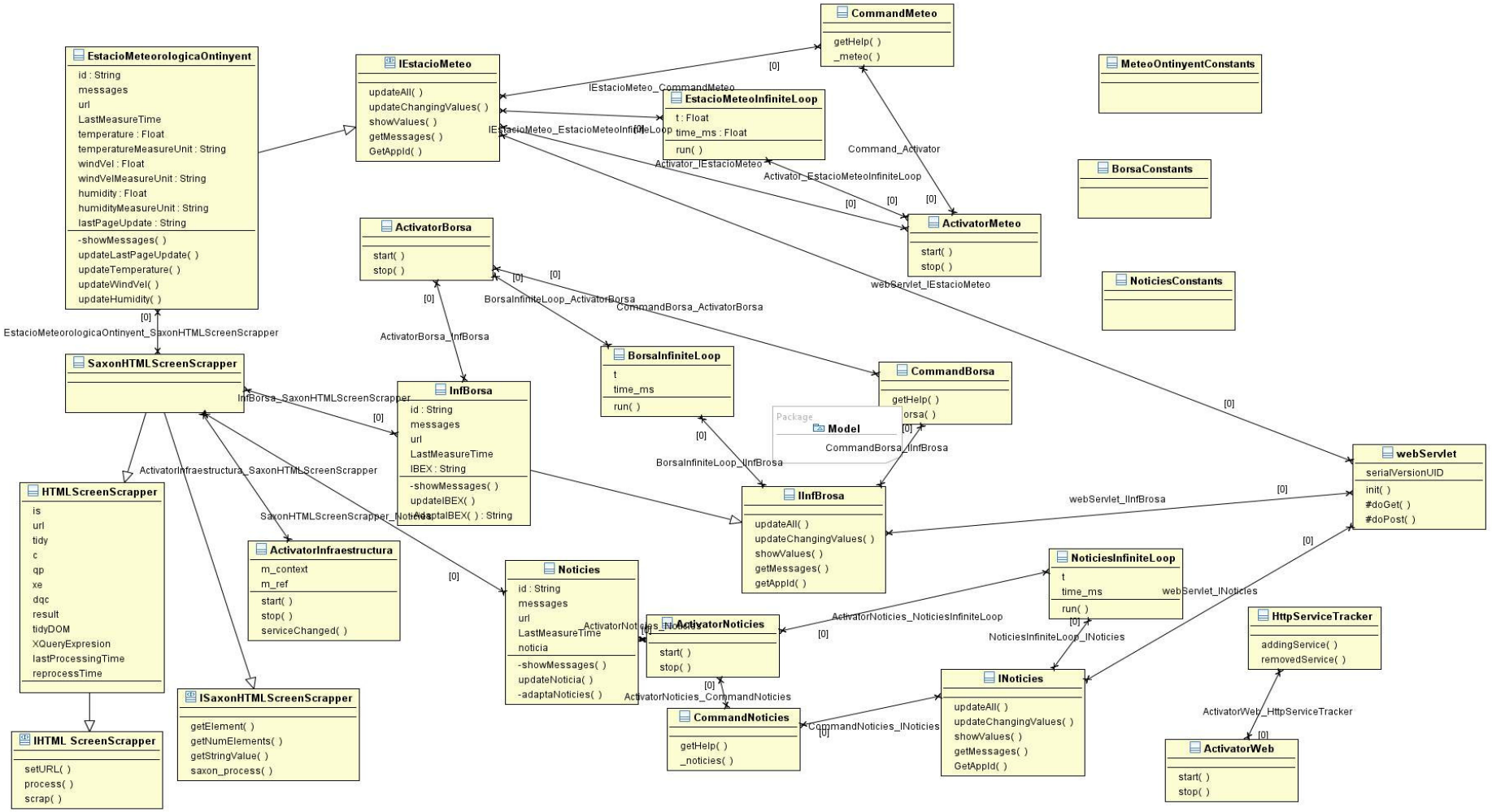


Imagen 14: Aproximación al diagrama de clases del proyecto final.

En un primer vistazo nos puede costar asimilar todo el contenido del diagrama de clases final. Pero si miramos con un poco más de detalle nos vamos a dar cuenta pronto de que no es más que las diferentes clases que se han explicado en el contenido de cada bundle relacionadas entre ellas.

De este modo, se va a tratar de describir una mínima trazabilidad para comprender mejor el funcionamiento del proyecto.

Una vez OSGI inicia todos los bundles, empezará la comunicación entre ellos. Primero será el bundle WEB mediante la clase *webServlet* quien se encargue, cuando haya sido registrado el Servlet, de solicitar, por ejemplo, la predicción meteorológica.

Desde este punto del diagrama vemos como la clase *EstacioMeteoInfiniteLoop* solicitará cada cierto tiempo la ejecución de las consultas que contenga la clase *OntinyentMeteoConstants* mediante la clase *EstacioMeteorologicaOntinyent*.

Ésta última, conectará directamente con el bundle infraestructura, más concretamente con la clase *SaxonHTMLScreenScrapper* que ya se encargará de gestionar las consultas y almacenarlas en la lista de mensajes.

Por último, estos mensajes serán devueltos al bundle WEB que desde el método doGet() se mostraran al usuario mediante la página web creada de forma dinámica.

Pensando en el caso de que el diagrama no haya quedado lo suficientemente explicado y claro, presentamos esta nueva versión del diagrama de clases final (Imagen 15), en el que hemos marcado el borde de cada clase con un color a razón de cada bundle. De este manera se verá mucho mejor la relación e interconectividad que existe entre ellos.

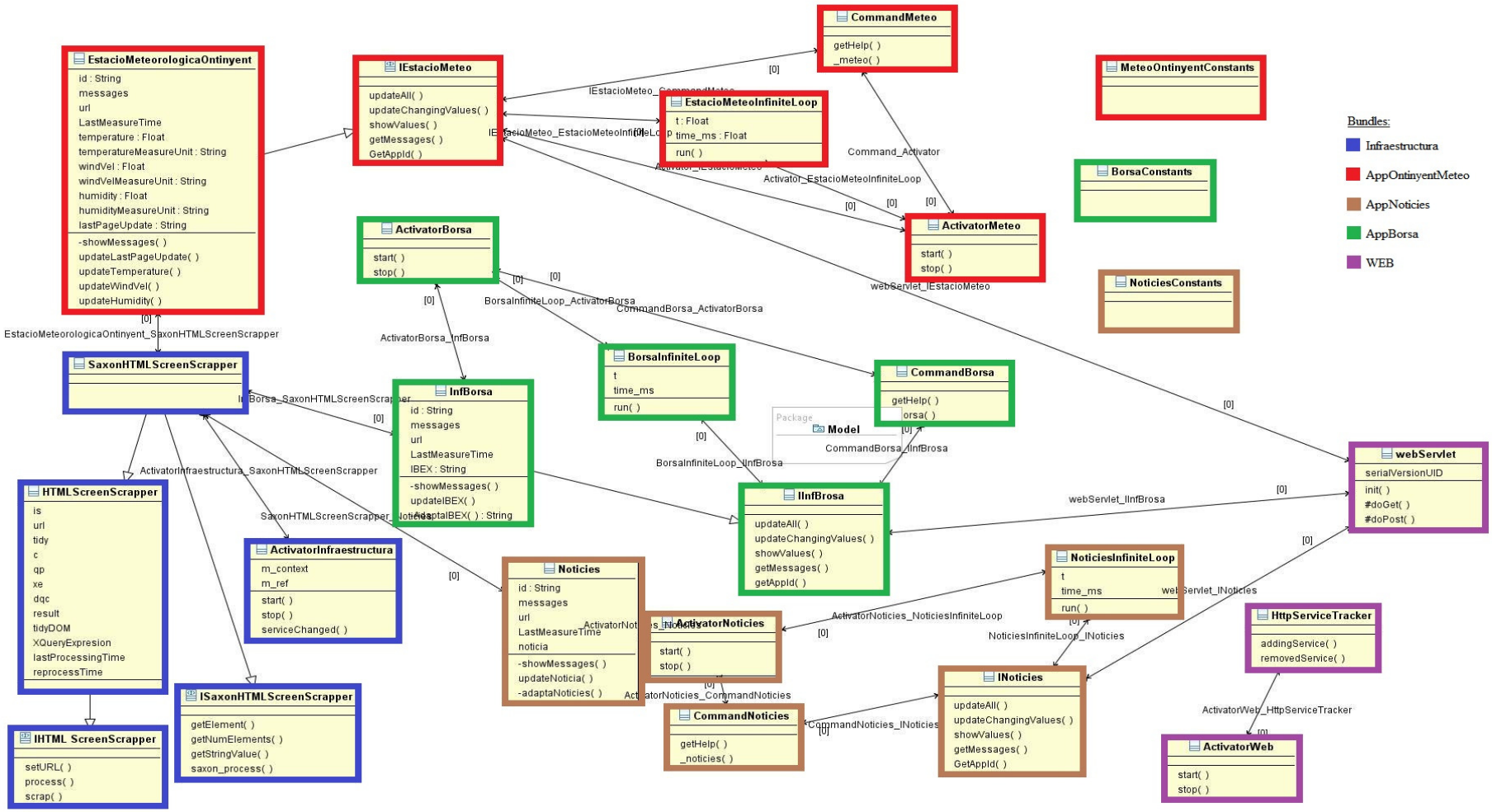


Imagen 15: Aproximación al diagrama de clases del proyecto final diferenciando los distintos bundles por colores.

5.- Implementación

Una vez ha quedado clara cual es el diseño de la aplicación y la funcionalidad que debe tener cada una de sus partes, veamos su implementación. Para ello se va a proceder de un modo similar al del apartado anterior, es decir, vamos a ver como implementa su funcionalidad cada uno de los bundles, o mejor dicho de cada uno de los niveles de bundles (infraestructura, aplicación y web).

Se va a realizar gran hincapié en la muestra de parte del código puesto que pese a ser de poca extensión considero que es fundamental exponer de manera clara y transparente como poder implementar esta técnica de Screen-Scrapping.

5.1 Infraestructura

Este paquete consta de un total de 5 clases y como ya se ha comentado su funcionalidad es tratar el código de las páginas web para generar un DOM bien formado sobre el que poder ejecutar las diferentes consultas XQuery.

Activator

Esta clase implementa la interfaz *BundleActivator* que tiene dos métodos, start() y stop(), en los que se indican las acciones que se deben realizar al iniciar o parar el bundle. En el método start() también se incluye el código encargado de registrar en el entorno los servicios que implementa el Bundle.

```
public void start(BundleContext context) throws Exception
{
	System.out.println("Start Infraestructura bundle!!");

	m_context = context;
	m_context.addServiceListener(this, "(objectClass="+
		SaxonHTMLScreenScrapper.class.getName()+")");
	m_ref =	m_context.getServiceReference(
		SaxonHTMLScreenScrapper.class.getName());

	if(m_ref != null)
		ss = (SaxonHTMLScreenScrapper)m_context.getService(m_ref);

	System.out.println("Service registered: Infraestructura");
}

public void stop(BundleContext context) throws Exception
{
	System.out.println("Stop Infraestructura bundle!!");
}

public void serviceChanged(ServiceEvent event)
{
	String[] objectclass =
(String[])event.getServiceReference().getProperty("objectclass");

	if(event.getType()==ServiceEvent.REGISTERED)
	{
		if(m_ref==null)
		{
			m_ref = event.getServiceReference();
			ss =
(SaxonHTMLScreenScrapper)m_context.getService(m_ref);
		}
	}
	else if(event.getType()==ServiceEvent.UNREGISTERING)
	{
		if(event.getServiceReference()==m_ref)
		{
			m_context.ungetService(m_ref);
			m_ref=null;
			ss=null;
			m_ref=m_context.getServiceReference(
SaxonHTMLScreenScrapper.class.getName());

			if(m_ref != null)
		ss=SaxonHTMLScreenScrapper)m_context.getService(m_ref);
		}
	}

}
```

HTMLScreenScrapper

Esta clase implementa la interfaz *IHTMLScreenScrapper* que contiene los métodos setURL() y scrap(). Estos métodos permiten modificar la URL que tengamos almacenada y ejecutar las XQuery, respectivamente.

Además implementa un nuevo método process() que será quien procese la página correspondiente a la URL que se le pase como parámetro y generar un DOM bien formado.

```
public void setURL(URL url) { this.url = url;}

private void process(URL url)
{
            //Evita reprocesar la pagina web, i por tanto utilizar la
que tingo en cache

            Boolean process = false;
            if ( url == null )
                  return;
            Date now = Calendar.getInstance().getTime();

            //Si todavia no la he procesado, lo tengo que hacer
            if ( this.lastProcessingTime == null )
                  process = true;
            else {
                  //si ya la he procesado, compruebo si han pasado
MeteoOntinyentConstants.REPROCESS_WEB_PAGE_INTERVAL segundos

                  long millis =
now.compareTo(this.lastProcessingTime);

                  process = ( (now.getTime() -
this.lastProcessingTime.getTime()) > this.reprocess_time);

            }

            //Si antes no habia URL especificada, la proceso
            if ( this.url != url )
                  process = true;

            if ( process ) {
                  this.lastProcessingTime = now;
                  try {
                        is = new InputStreamReader(url.openStream());
                  } catch (IOException e) {
                        e.printStackTrace();
                  }

                  tidyDOM = tidy.parseDOM(is, null);

                  dqc.setContextNode(new DocumentWrapper(tidyDOM,
url.toString(), c));
            }
      }

public String scrap(String xquery) {

      if ( this.url == null )
            return "";

      this.process(url);
      try {
            xe = qp.compileQuery(xquery);
      } catch (XPathException e) {
            e.printStackTrace();
```

```
        }

        try {
                result = xe.evaluate(dqc);
        } catch (XPathException e) {
                e.printStackTrace();
        }

        if ( result == null )
                return "";

        String output = "";
        String value_sep = " ";
        Iterator<TinyElementImpl> i = result.iterator();
        TinyElementImpl el = null;
        while ( i.hasNext() ) {
                el = i.next();
                if ( !output.equals("")) output += value_sep;
                output += el.getStringValue();
        }

        return output;
}

public String scrap(URL url, String xquery)
{
        this.setURL(url);

        return this.scrap(xquery);

}
```

SaxonHTMLScreenScrapper

Esta clase extiende de *HTMLScreenScrapper* e implementa la interfaz *ISaxonHTMLScreenScrapper*, y va a ser, llamémosle así, la puerta de enlace a los bundle de aplicación.

Su función, a parte de la de pasarela entre las diferentes capas de bundle, es la de permitir navegar por los DOM que se generan al procesar las URL y obtener de ellos la información contenida en determinados nodos que sean de nuestro interés.

```
public TinyElementImpl getElement(int i)
{
        if ( this.result != null && this.result.size() >= i )
                return this.result.get(i);
        return null;
}

public int getNumElements()
{
        return this.result.size();
}

public String getStringValue(int i)
{
        TinyElementImpl e = this.getElement(i);
        if ( e != null )
```

```
            return e.getStringValue();
      return "";
}

public List<TinyElementImpl> saxon_process(URL url, String xquery)
{
      this.scrap(url, xquery);
      return this.result;

}
```

5.2 AppBorsa, AppNoticies y AppOntinyentMeteo

Como ya se ha comentado anteriormente, la similitud entre estos tres bundles es considerable, y, por tanto, se va a explicar la implementación del bundle AppBorsa. Por lo que cualquier explicación sobre éste es totalmente extensible a los otros dos.

La principal tarea de este bundle consiste en solicitar al bundle infraestructura la ejecución de las XQuery que contiene en la clase *BorsaConstants* y almacenar el resultado en la lista de mensajes que serán después mostrados al usuario.

Activator

La similitud de la clase *Activator* de este bundle con la del anterior es elevada, por lo que no se va a comentar en exceso. Símplemente señalar que se desde aquí es desde donde se ejecutan el nuevo hilo de ejecución que mantendrá la información recopilada de la URL de manera actualizada, realizando cada periodo de tiempo la consulta sobre la página.

```
public void start(BundleContext context) throws Exception
{
      System.out.println("Start InfBorsa bundle!!");

      cmdB = new Command(borsa);
      context.registerService(CommandProvider.class.getName(), cmdB,
null);
      borsa = new InfBorsa();

      loopB = new BorsaInfiniteLoop(borsa);
      new Thread(loopB).start();

      System.out.println("Service registered: AppBorsa");
}

public void stop(BundleContext context) throws Exception { }
```

BorsaConstants

Hasta el momento, se había considerado BorsaConstants como una clase, cuando en realidad se ha implementado mediante una interfaz que almacena una serie de valores contantes necesarios para el funcionamiento de la aplicación. Estos valores podrían haber sido solicitados al usuario de la aplicación mediante algún tipo de interfaz gráfica

o cualquier otro modo. Se optó por esta solución por comodidad en la implementación y por no tratarse de ningún requisito del proyecto.

Concretamente, aquí se va a almacenar: un identificador del bundle, la URL de la página web que queremos tratar, el tiempo que queremos estar utilizando la página que hay almacenada en caché y la XQuery que se quiere lanzar.

```
public interface BorsaConstants
{
	public final String id = "Informació borsa";
	public final String URL =
"http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.htm";

	public final long REPROCESS_WEB_PAGE_INTERVAL = 1; // Num.
segundos durante el cual no reprocesare la pagina web i utilizare la
que tengo en cache

	public final String XQUERY_IBEX35 = "for $d in
//td[contains(text()[1], \" IBEX 35®\")] return
<span>{data($d/parent::tr/following-sibling::tr)}</span>";


}
```

BorsaInfiniteLoop

Esta clase permite definir un periodo de tiempo durante el cual no se actualizaran los valores obtenidos como resultado de la XQuery ejecutada en el bundle.

```
public void run() {

	int i=0;
	while ( i++ < t ) {
		b.updateAll();

		try {
			new Thread().sleep(time_ms);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

}
```

InfBorsa

Esta es la última clase que vamos a repasar del nivel de bundles de aplicación. Su tarea es

```
public InfBorsa()
{
	try {
		this.url = new URL(BorsaConstants.URL);
	} catch (MalformedURLException e)
	{
		e.printStackTrace();
```

```
        }

        this.ss = new SaxonHTMLScreenScrapper();
        this.ss.setURL(url);

}

[…]

public void updateAll() {

        this.messages.clear();
        this.LastMeasureTime = Calendar.getInstance().getTime();

        this.updateIBEX();
        this.showMessages();

}

public void updateIBEX()
{
        String sData;
        ss.scrap(BorsaConstants.XQUERY_IBEX35);
        sData = ss.getStringValue(0);

        String newIBEX = AdaptaIBEX(sData.toString());

        int r = newIBEX.compareTo(this.IBEX);

        if(r != 0)
                this.messages.add("Valor del IBEX 35: " +
newIBEX.toString());

        this.IBEX = newIBEX;
}

private String AdaptaIBEX(String ibex)
{
        String ibex_ok = "";
        int contador = 0, pos = 0;
        boolean seguir = true;

        for(int i = 0; i < ibex.length() && seguir; i++)
        {
                if(ibex.toCharArray()[i] == ',' && contador < 2)
                        contador++;
                else
                        if(contador == 2)
                        {
                                seguir = false;
                                pos = i + 2;
                        }
        }

        ibex = ibex.substring(pos);
        seguir = true;

        for(int i = 0; i < ibex.length() && seguir; i++)
        {
                if(ibex.toCharArray()[i] == ',')
                {
```

```
                    pos = i + 2;
                    seguir = false;
                }
        }

        ibex_ok = ibex.substring(0, pos+1);

        return ibex_ok;
}

public void updateChangingValues()
{
        this.messages.clear();
        this.LastMeasureTime = Calendar.getInstance().getTime();

        this.showMessages();
}

[…]
```

Una vez explicadas estas clases es podemos entender con mayor facilidad que para cualquier aplicación que deseemos crear de extracción de datos web utilizando ésta técnica, se hará de una forma muy parecida y utilizando clases muy semejantes a las del bundle AppBorsa.

5.3 WEB

El bundle WEB tiene un tamaño menor, solamente tres clases: *Activator*, *HttpServiceTracker* y *webServlet*. Y su función es la de coordinar las solicitudes de ejecución de las diferentes XQuery de las diferentes aplicaciones. Una vez hayan actuado dichas aplicaciones y se tenga toda la información, requerida por ellas, almacenada en los mensajes que se desean mostrar al usuario, se generará una página web dinámicamente (Servlet) en la que se muestre el contenido de toda la lista de mensajes.

Activator

Poca cosa más que no se haya comentado ya sobre el funcionamiento y funcionalidad de esta clase que es necesaria en cada bundle.

```
public void start(BundleContext context) throws Exception
{
        serviceTracker = new HttpServiceTracker(context);
        serviceTracker.open();
}

public void stop(BundleContext context) throws Exception {
        serviceTracker.close();
        serviceTracker = null;

}
```

HttpServiceTracker

Es necesaria para poder registrar el Servlet en el directorio que se desee y de esta manera permitir que el servidor Tomcat pueda procesar correctamente las peticiones y enviar las respuestas correspondientes.

```
public HttpServiceTracker(BundleContext context)
{
	super(context, HttpService.class.getName(),null);
}

public Object addingService(ServiceReference reference)
{
	HttpService httpService = (HttpService)
super.addingService(reference);

	if(httpService == null)
		return null;

	try{
		System.out.println("Registering servlet at /web");
		httpService.registerServlet("/web", new webServlet(),
null, null);
	}catch(Exception e){
		e.printStackTrace();
	}

	return httpService;
}

public void removedService(ServiceReference reference, Object
service){
	HttpService httpService = (HttpService) service;

	System.out.println("Unregistering /web");
	httpService.unregister("/web");


	super.removedService(reference, service);

}
```

webServlet

En esta clase tan solo se ha sobrecargado el método doGet() que se va a ocupar de solicitar la ejecución de las consultas de las diferentes aplicaciones y generar el contenido de la página HTML a partir de los mensajes devueltos por los bundle de aplicación.

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
	response.setContentType("text/html");
	ArrayList<String> content = null;
```

```
[…]

      try{
            bor = new InfBorsa();

            bor.updateAll();

            content = bor.getMessages();

            if ( content.size() == 0 )
                  return;

            Iterator<String> j = content.iterator();

            response.getWriter().write("<h2>" + bor.GetAppId()+
"</h2>");
            while (j.hasNext())
                  response.getWriter().write(j.next() + "<br>");

            response.getWriter().write("<br>");
      }catch(Exception e){
            response.getWriter().write("No funciona la aplicació de
borsa.");
      }

[…]

}
```

6.- Caso de estudio

Una vez ya hemos explicado cual es el funcionamiento de nuestro proyecto, así como se ha procedido en su diseño e implementación, veamos cual es el aspecto final del mismo, qué se ha desarrollado y cual es el resultado de su ejecución.

Tal y como adelantamos en apartados anteriores, la finalidad de este proyecto es el estudio de la extracción de datos web usando técnicas de Screen-scraping. Y como ejemplos, se han implementado una serie de bundles de OSGI, en concreto tres.

Como ya sabemos, los ejemplos desarrollados son:

- Información meteorológica de la ciudad de Ontinyent.
- Valor actual del IBEX35.
- Noticias de última hora.

Los resultados de la aplicación dependerán del contenido de las páginas web de las cuales deseamos obtener la información. Las URL referentes a los ejemplos comentados son:

- http://www.meteorologiajaumeprimer.com/meteo/inicicentre2.php
- http://www.bolsamadrid.es/esp/mercados/acciones/accind1_1.htm
- http://www.rtve.es/noticias/

Veamos ahora el contenido de las distintas direcciones de las que vamos a extraer la información, y así conocer cual debe ser el resultado esperado de la ejecución de la aplicación.

Por lo que respecta a la Información meteorológica de la ciudad de Ontinyent, deseamos conocer: la última actualización de la página web, la temperatura, la velocidad del viento y la humedad, estos valores aparecen marcados en rojo en la Imagen 16. Como aclaración resaltar que, a parte de estos datos, podríamos haber seleccionado otros de los muchos que salen en esta web.

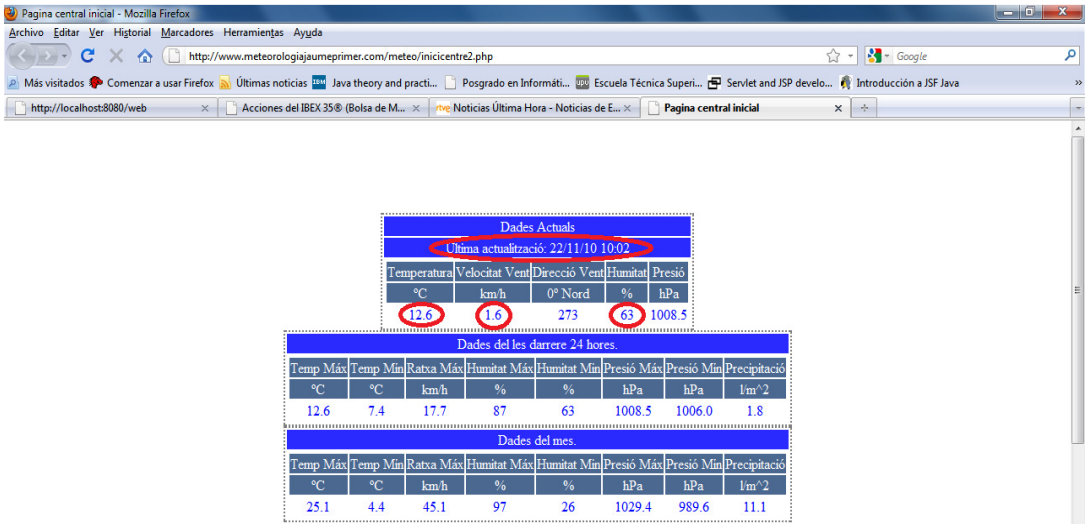


Imagen 16: Contenido de la página web de información meteorológica.

En la imagen aparecen los valores esperados tras la ejecución, estos deberían ser: 22/11/10 10:02, 12.6, 1.6, 63.0.

Por otro lado, el aspecto de la página de la que se va a extraer el valor del índice de referencia de la bolsa española (Imagen 17), es el siguiente:

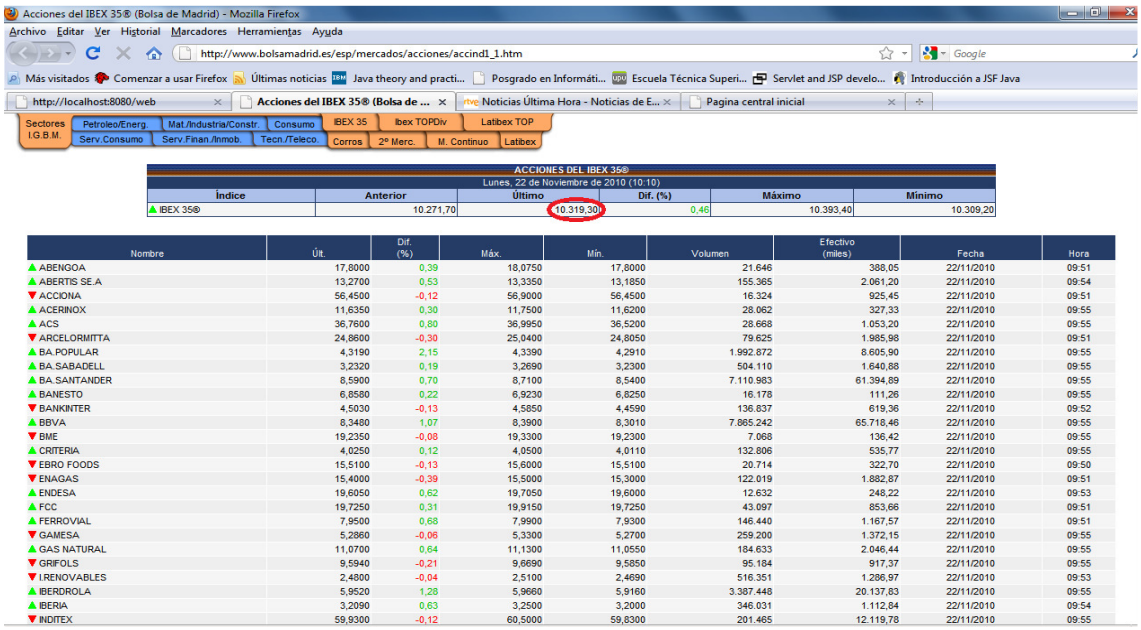


Imagen 17: Contenido de la página web que contiene el valor del IBEX35.

Según indica el último valor del IBEX35, el resultado que nos debe devolver la aplicación es: 10.319,30.

Y por último, conozcamos el contenido de la web de RTVE (Imagen 18), de donde se va a capturar las noticias de última hora que ahí se muestran:

Imagen 18: Contenido de la página web de noticias.

En la imagen 15 se recuadran algunas de las noticias que deberán ser mostradas tras la ejecución del programa.

Una vez hemos elegido y conocemos cuales son los valores que vamos a extraer en los ejemplos implementados, comprobemos que realmente el resultado es el esperado.

Cabe destacar, que en una primera versión del proyecto, la información devuelta por las XQuery no se mostraba mediante un Servlet, sino que se mostraba el texto mediante la consola del propio Eclipse, mediante el cual se ha desarrollado la totalidad del proyecto.

En algunos detalles es un tanto diferente la información mostrada en el Servlet, que la que se muestra en la consola. El motivo es que para asegurarnos de la correcta puesta en marcha de cada bundle, se mostraba por consola un mensaje de confirmación, o por el contrario, de error. En el Servlet, para reducir el contenido del mismo y al mismo tiempo transferirle una interfaz con el usuario más clara y directa, ese tipo de mensajes no se han mostrado.

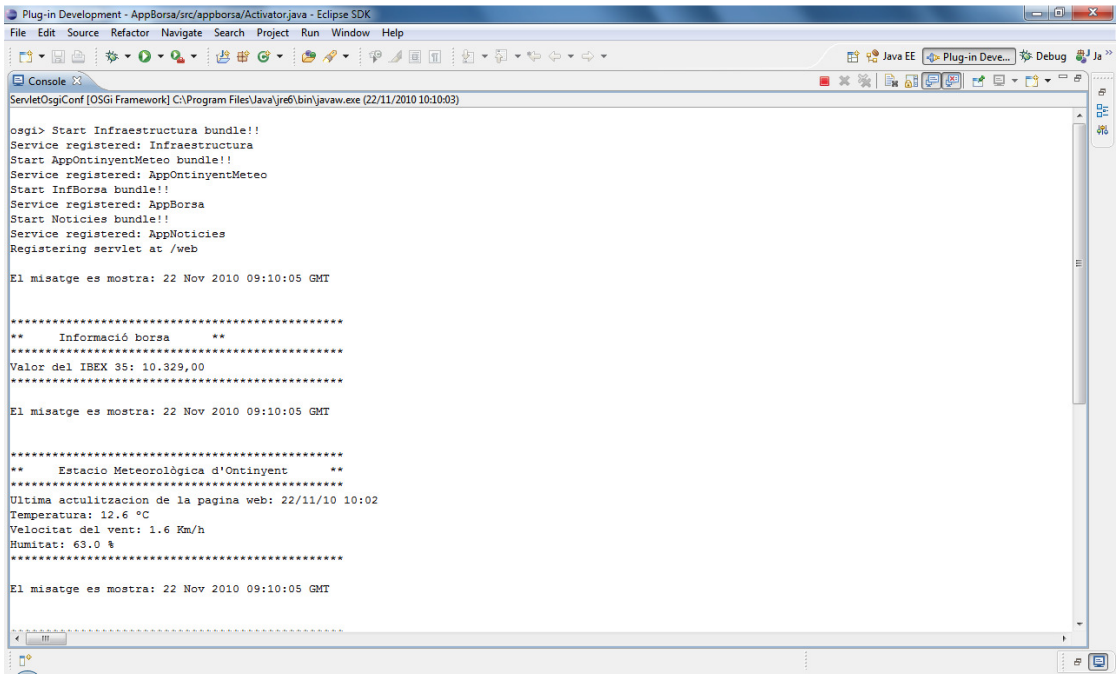


Imagen 19: Resultado de la ejecución. Mensajes mostrados en la consola de Eclipse.

En la Imagen 19, podemos comprobar como se nos indica que los bundles Infraestructura, AppOntinyentMeteo, AppBorsa, AppNoticies y Web, han sido iniciados sin ningún error. A parte, también se puede ver el resultado, mostrado como texto, de las XQuery.

Si nos fijamos, notamos que el valor del IBEX35 que se muestra en la consola, no es el mismo que se había señalado en la captura de la página web, esto se debe a la constante fluctuación de este valor. Se ha comprobado que la ejecución de la ejecución muestra el valor correcto, de hecho, veremos a continuación como el Servlet si que muestra el mismo valor que la captura de la web. Este pequeño desajuste nos sirve para demostrar que la aplicación varía los valores a medida que se actualizan los datos contenidos en las URL sobre las que lanzamos las consultas. Pues en el tiempo en que se han ido tomando las capturas de pantalla para la mejor compresión de lo aquí explicado, los valores del IBEX35 se han actualizado y el programa responde con éxito a este hecho.

Solamente nos falta por ver cual es el aspecto de la web que generamos gracias a Servlet, y que puede que sea la parte que más interese al posible usuario de este software.

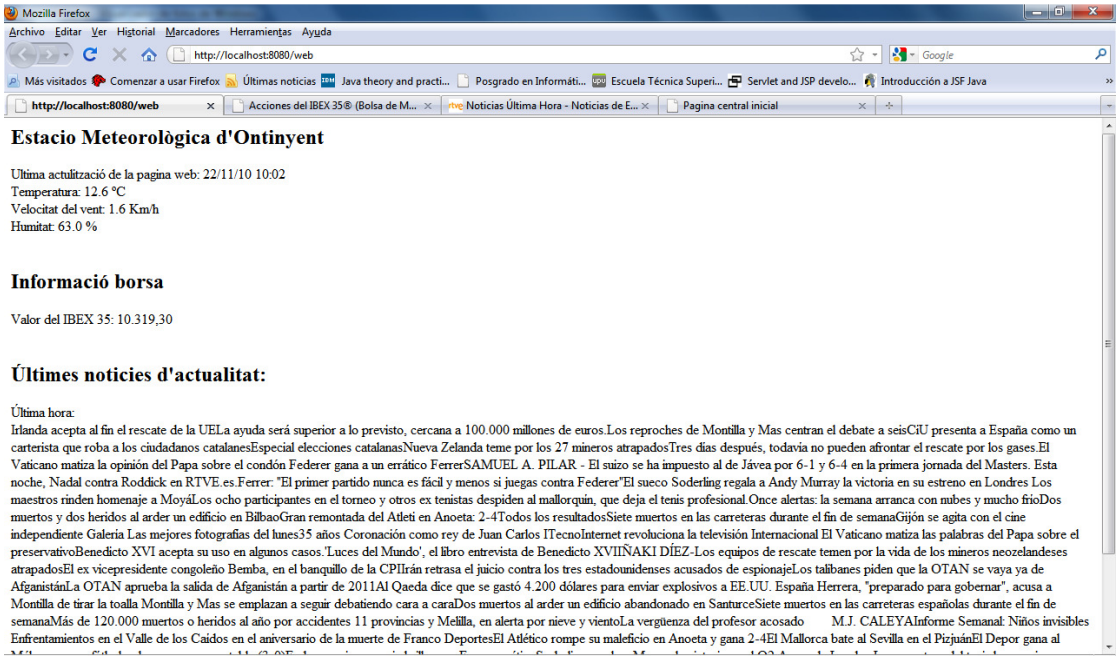


Imagen 20: Resultado de la ejecución. Contenido de la web generada por el Servlet.

En la Imagen 20, sí que se puede comprobar como los valores que esperábamos son los correctos: Ultima actualización de la pagina web: 22/11/10 10:02, Temperatura: 12.6ºC, Velocidad del viento: 1.6 Km/h, Humedad: 63.0 %; Valor del IBEX 35: 10.319,30; y las noticias que se muestran en la web de RTVE.

7.- Conclusiones

Si, una vez finalizado todo el trabajo de este proyecto final de carrera, nos paramos a pensar qué conclusiones podemos sacar de todo el proceso que se ha seguido, desde el planteamiento inicial hasta el final del desarrollo, es evidente que, a diferencia de otros proyectos final de carrera, el principal objetivo de éste no ha sido la implementación de un sistema informático con una funcionalidad extensa y dirigido a un usuario final que pueda utilizar la aplicación frecuentemente, sino que el objetivo más importante es el aprendizaje de la técnica de Screen-scraping que se ha ido explicando a lo largo del documento.

Como ya se ha explicado, el proceso de aprendizaje ha marcado fuertemente el desarrollo del proyecto, es decir, se ha ido implementando a medida que se iba aprendiendo sobre el funcionamiento de las XQuery. Por lo que en un principio se trató de implementar un ejemplo sencillo que sirviese de aprendizaje y una vez se había, más o menos, dominado la técnica, se estudiaron diferentes diseños que permitiesen que nuestra aplicación funcione de la manera más eficiente posible y reutilizando la mayor cantidad de código posible. Una vez ya se tuvo el diseño claro, se implementaron el resto de los ejemplos que ya se han mostrado y comentado.

Se ha conseguido desarrollar una aplicación capaz de extraer determinada información de distintas páginas web y dejar parte del trabajo listo para que puedan ser más aplicaciones las que puedan extraer información de otras páginas, o que la información que se haya extraído, en lugar de ser mostrada mediante un Servlet, pueda

ser aprovechada por otros bundles para otro propósito totalmente distinto, por ejemplo, para facilitar parte de la gestión de una vivienda inteligente.

Básicamente, mi labor como futuro Ingeniero Técnico en Informática de Sistemas, ha sido el aprendizaje de, para mi, nuevos lenguajes de consultas (XQuery), nuevas tecnologías (OSGI, Servlet) y nuevos modos de trabajo (se plantea un problema que debe resolver uno mismo), así como todas las labores de análisis y diseño, que después de el aprendizaje, han sido las partes en las que más tiempo se ha invertido, tal y como se debe actuar según las nuevas metodologías de desarrollo software.

Apéndice A1
Manual de usuario.

Como ya se ha comentado, uno de los objetivos del diseño que se ha elegido para este proyecto, es que el mismo sirva como base para poder crear otras aplicaciones que utilicen parte del código ya implementado.

Vamos a explicar brevemente como se debería proceder para poder utilizar esta característica.

Primer paso: Instalación de Eclipse.

El primer paso para poder utilizar el proyecto desarrollado es la instalación o adquisición de la herramienta Eclipse. Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma, es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Podemos conseguir la herramienta de forma gratuita en http://www.eclipse.org/. De las diferentes versiones que encontraremos de Eclipse, escogeremos la versión GALILEO.

Segundo paso: Seleccionar el Workspace.

Una vez se ha adquirido la herramienta, se deberá seleccionar el directorio "Workspace", donde escogeremos la carpeta "workspace_PFC", que es donde se encuentra el contenido del proyecto.

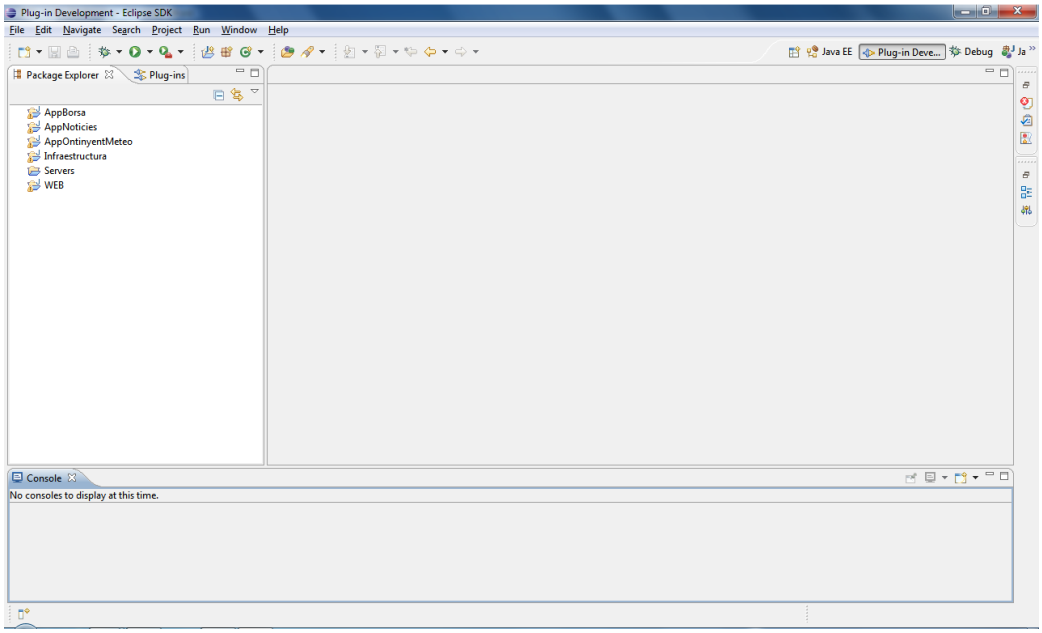


Imagen 21: Aspecto de Eclipse una vez cargado el Workspace "workspace_PFC"

Una vez cargado correctamente el Workspace, comprobaremos que se han cargado correctamente las seis carpetas de proyectos que contiene. Concretamente una por cada bundle más otra adicional que es necesaria para que funcione correctamente el servidor Tomcat.

Tercer paso: Creación de un nuevo bundle.

A continuación se especifican los pasos para crear un proyecto Java correspondiente a la estructura de un bundle OSGi con Eclipse.

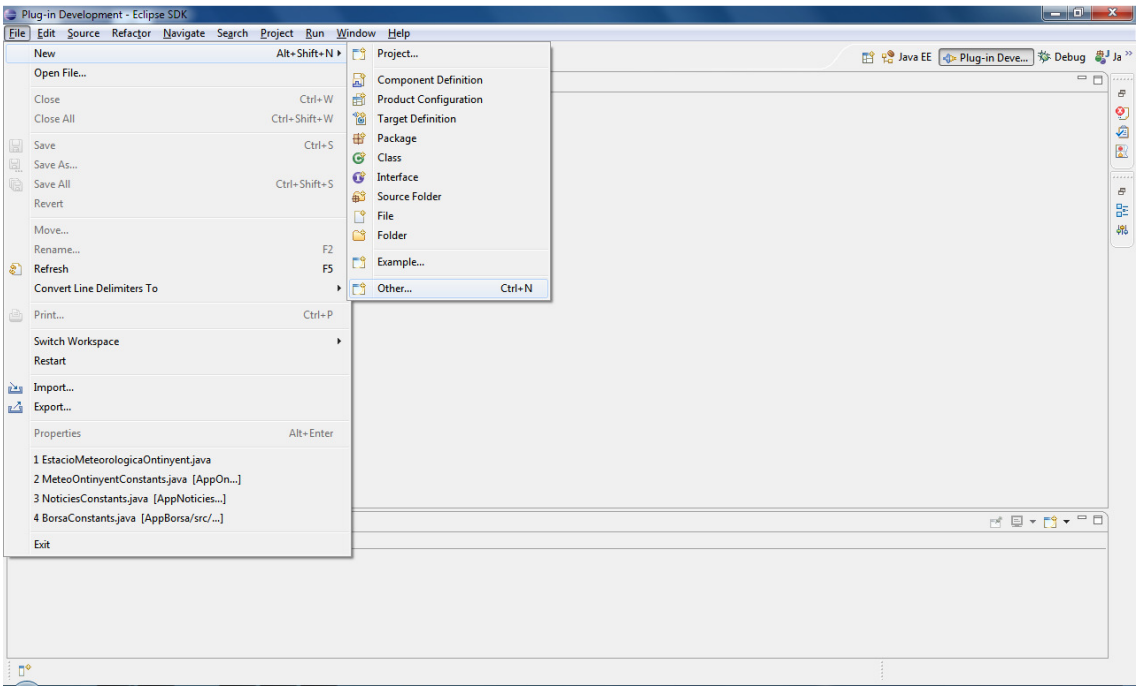


Imagen 22: Primer paso para crear un bundle OSGI con Eclipse.

En el siguiente menú elegiremos la opción Plug-in Project para crear un nuevo bundle.

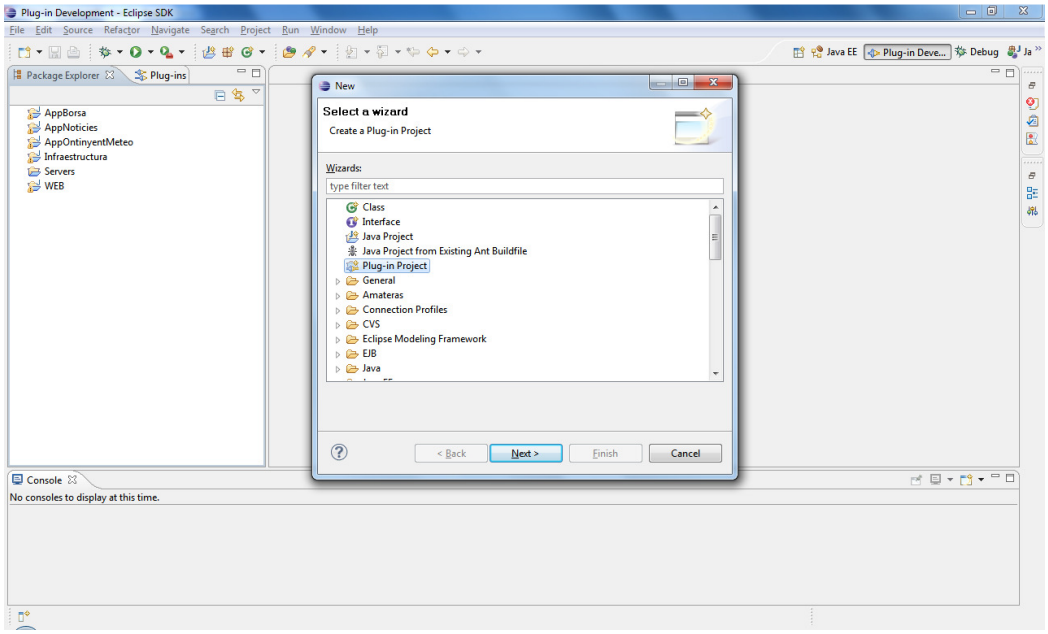


Imagen 23: Segundo paso para crear un bundle OSGI con Eclipse

En este paso, debemos darle un nombre al proyecto, o lo que es lo mismo, al nuevo bundle que estamos creando. Al mismo tiempo debemos seleccionar el Framework estándar de OSGI.

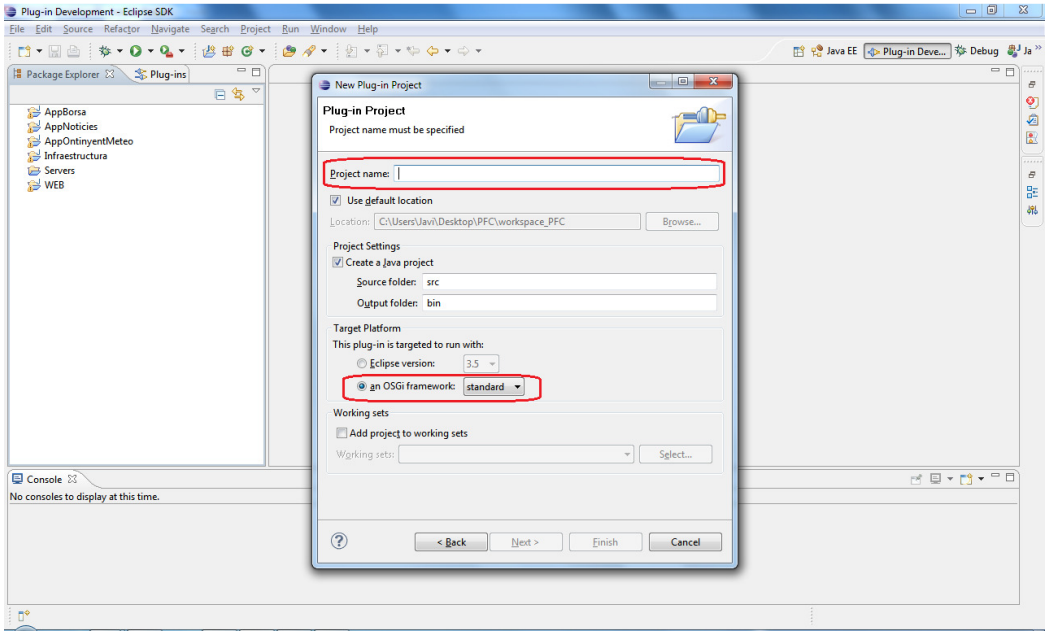


Imagen 24: Tercer paso para crear un bundle OSGI con Eclipse

En el siguiente paso, nos aseguraremos que como entorno de ejecución no haya ningún entorno seleccionado, y además marcaremos el checkbox para que se cree automáticamente la clase Activator del bundle.

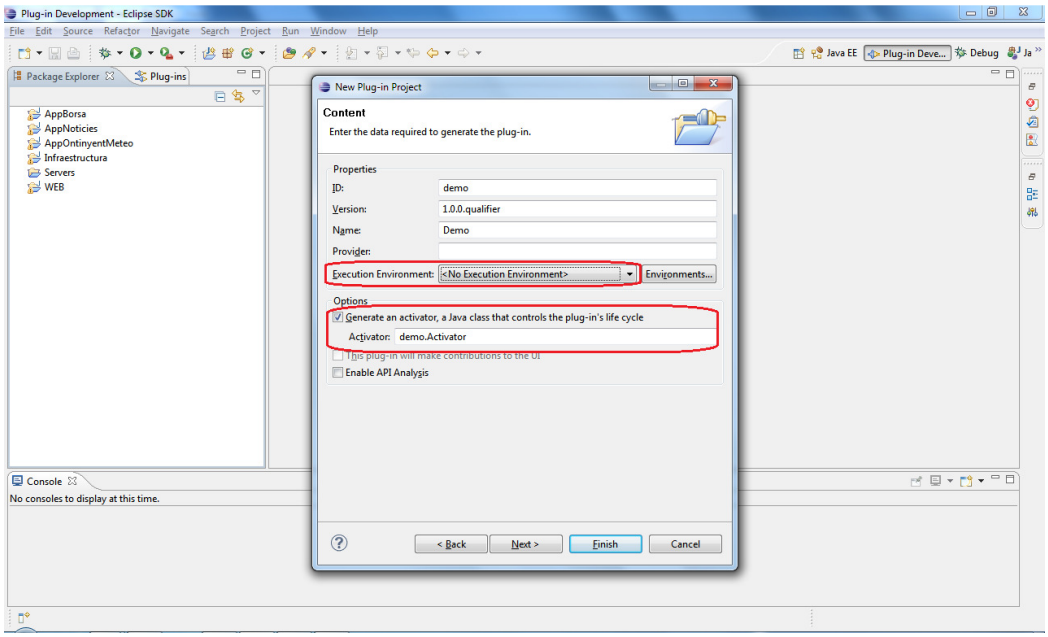


Imagen 25: Cuarto paso para crear un bundle OSGI con Eclipse

Por último, descartaremos que se cree el plug-in usando otras plantillas, y de esta manera ya tendremos creado nuestro nuevo bundle OSGI.

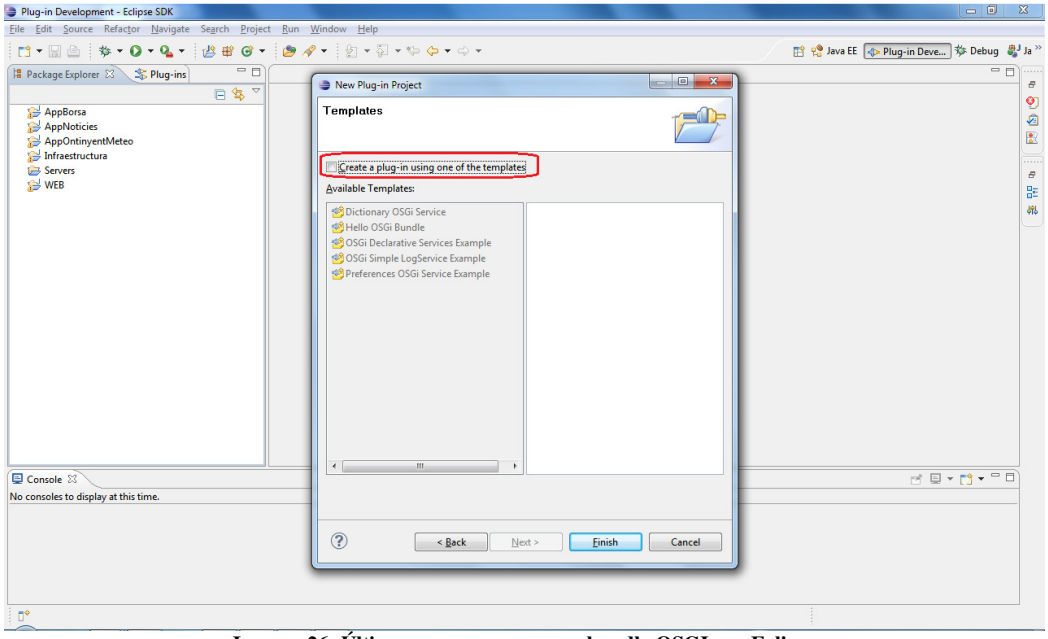


Imagen 26: Último paso para crear un bundle OSGI con Eclipse

Cuarto paso: Configuración del archivo Manifest.MF

Overview

La primera pestaña que vemos en el Manifest.MF es la de Overview:

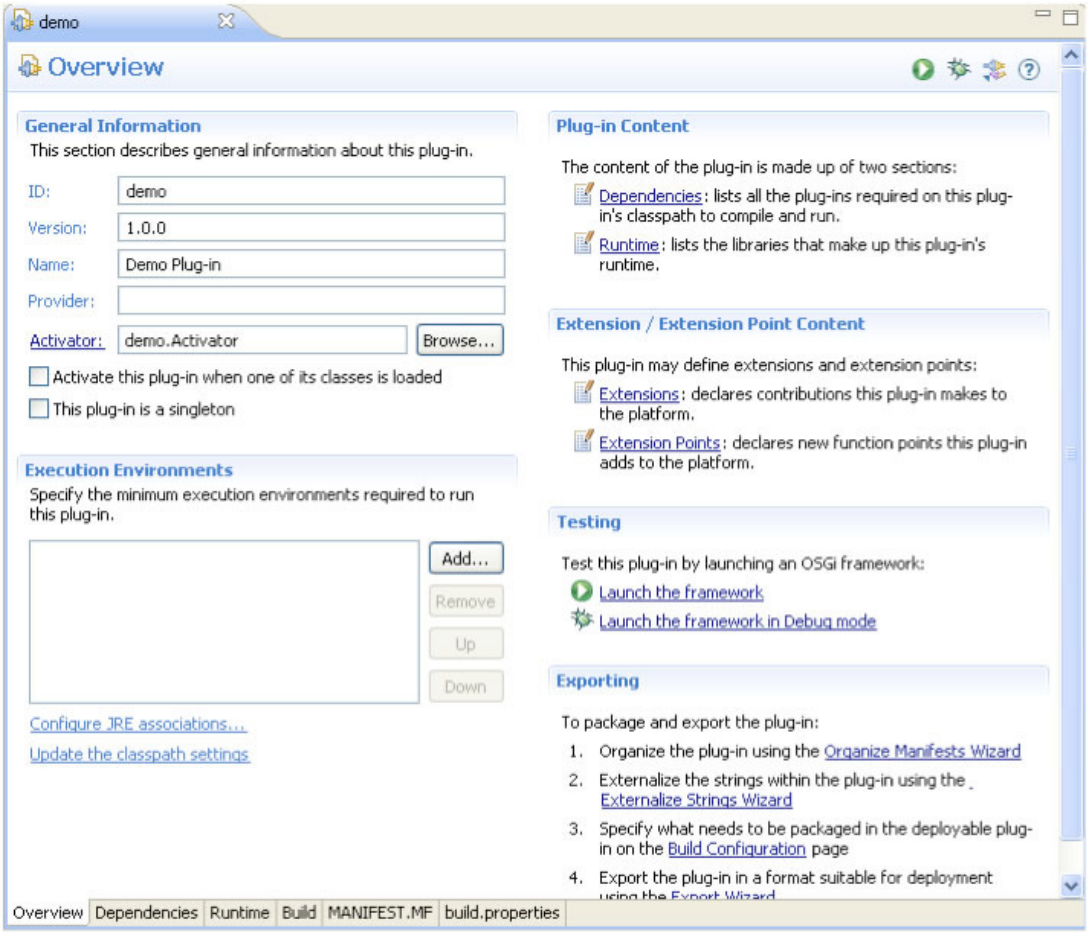


Imagen 27: Primera vista del Manifest.MF

En esta pestaña podemos ver la información general del bundle en la sección General Information. Esta información identificará después el bundle en el entorno de ejecución.

Dependencies

En la pestaña de dependencias, gestionaremos los paquetes que el bundle necesite para un correcto funcionamiento, como por ejemplo el framework OSGi, las interfaces, las constantes o bundles como el de Infraestructura. Para añadir paquetes solo debemos pulsar el botón Add… en la sección Imported Packages. El filtro resulta especialmente útil para buscar el paquete deseado.

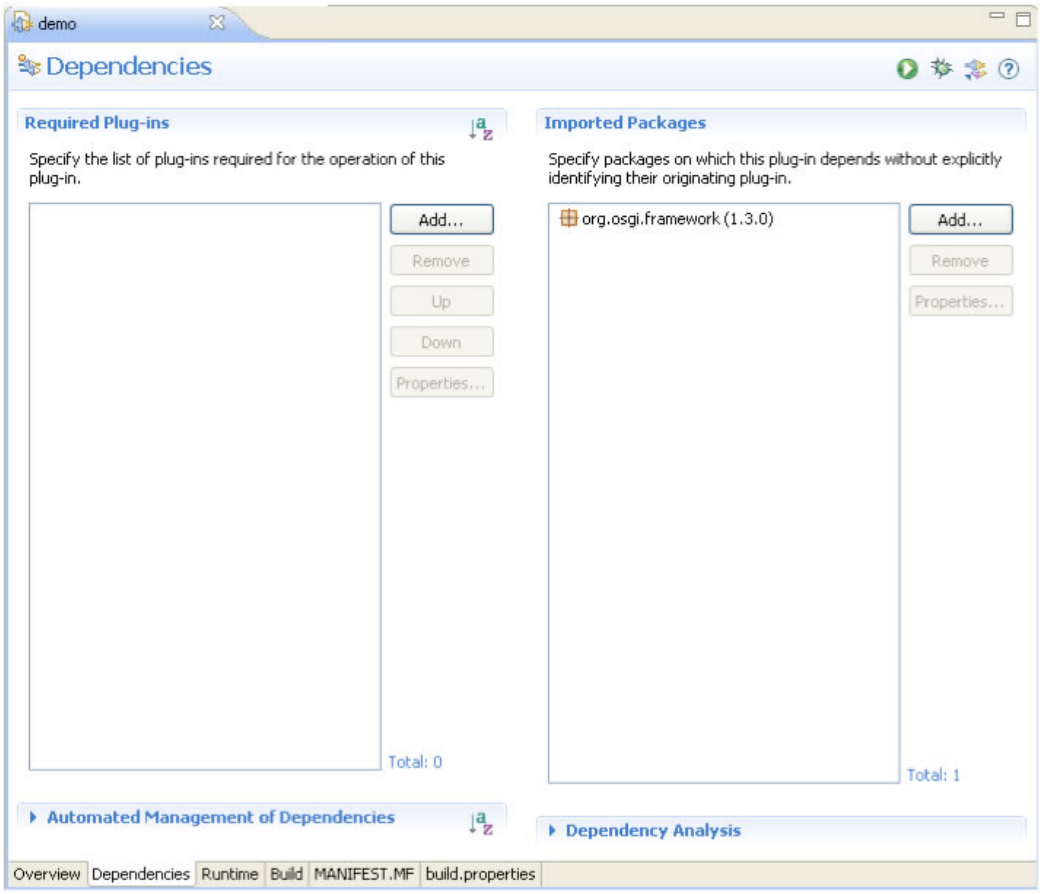
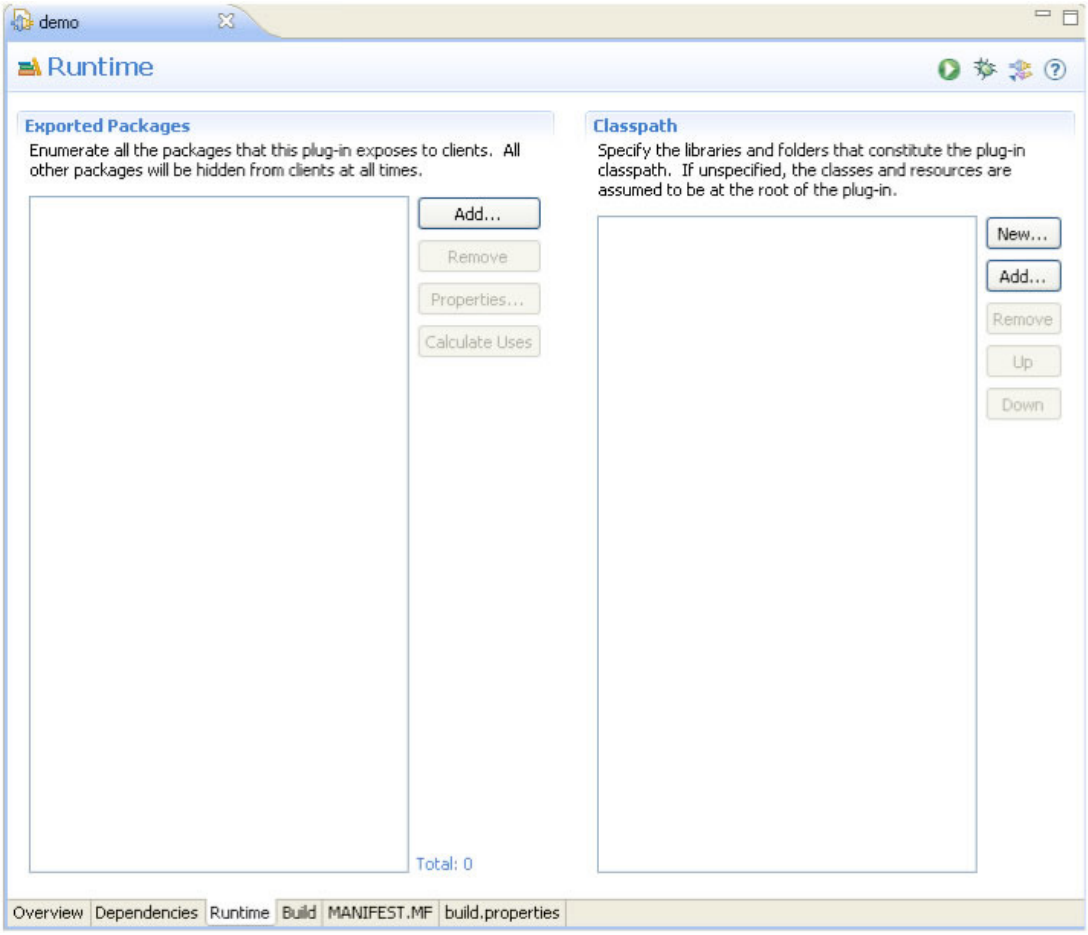


Imagen 28: Pestaña Dependencies del Manifest.MF

Runtime

De la pestaña runtime cabe destacar la sección Exported Packages en la que indicaremos que paquetes de nuestro bundle queremos que sean visibles para el resto del registro de OSGi, y de este modo poder ser usados a su vez por otros bundles.



29: Pestaña Runtime del Manifest.MF

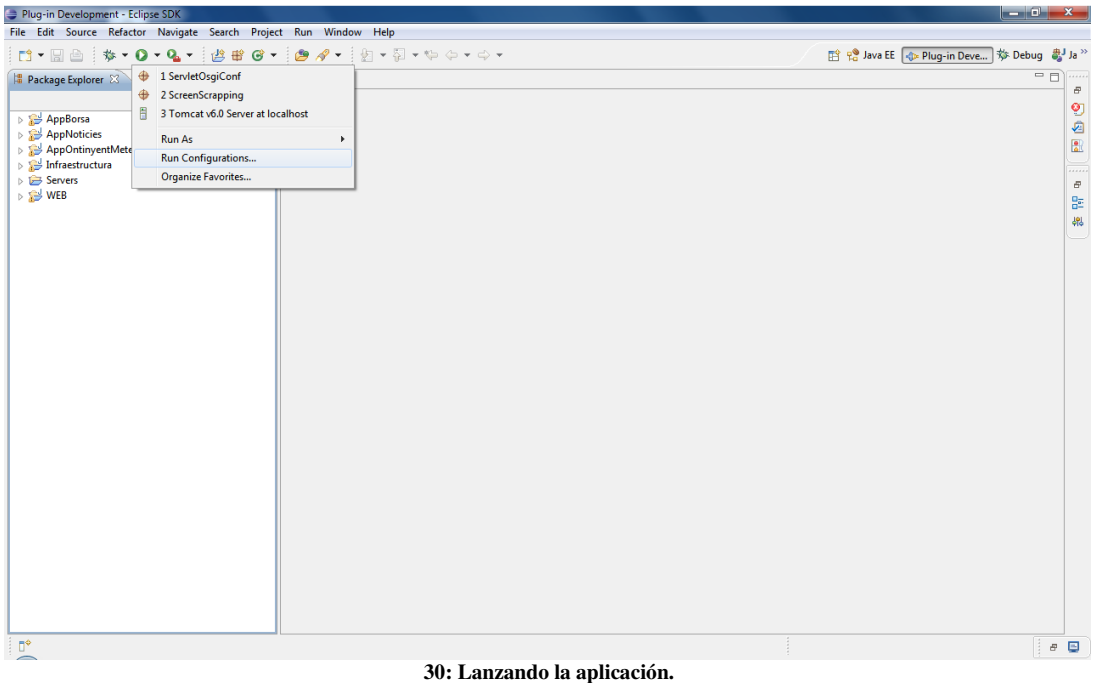
MANIFEST.MF

La pestaña MANIFEST.MF nos muestra el archivo en modo textual. No es recomendable manipular el archivo en este modo ya que podría causar fallos en la configuración.

Quinto paso: implementación del nuevo bundle.

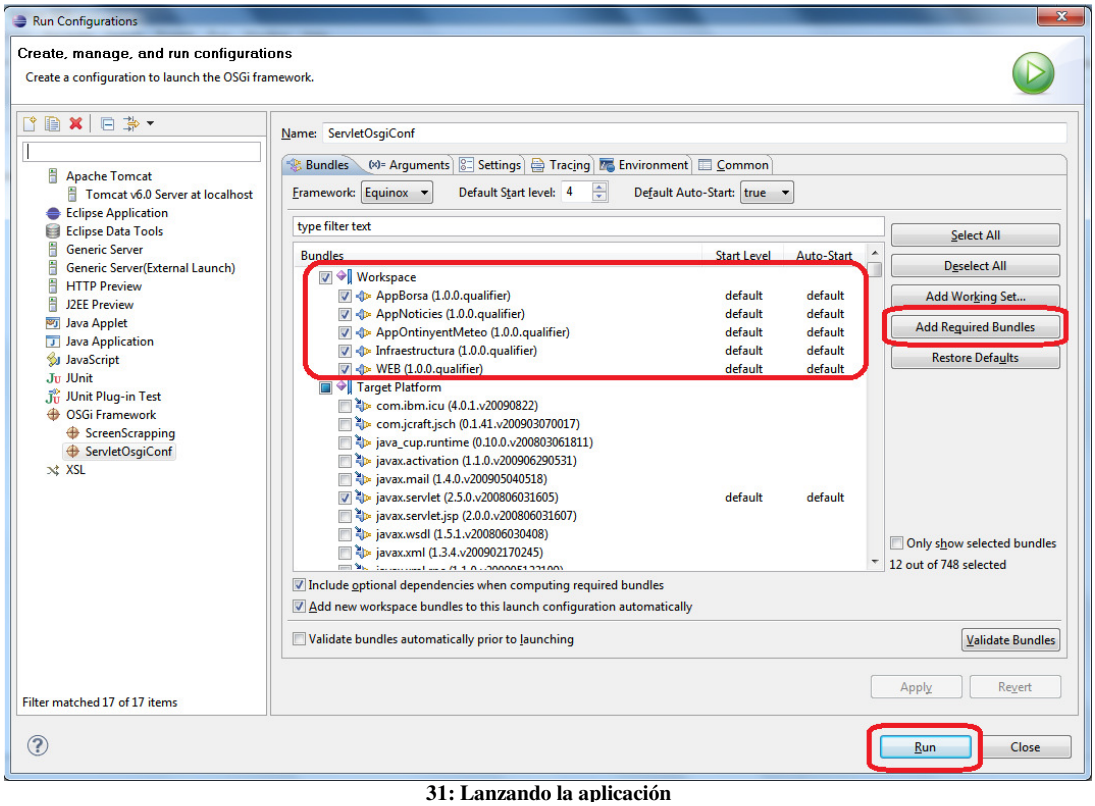
No por se el último paso debe ser el menos importante. En él, introduciremos todo el código necesario para el correcto funcionamiento, siguiendo el mismo patrón de implementación que se ha visto en apartados anteriores.

Una vez implementado, únicamente nos faltará lanzar la aplicación, para ello accedemos a:



30: Lanzando la aplicación.

Y nos aparecerá la siguiente ventana:



31: Lanzando la aplicación

Aquí es importante seleccionar todos los bundles que se han incluido en el Workspace, así como, hacer click en el botón “Add Required Bundles” para seleccionar todos los bundles necesarios para la correcta ejecución de la nueva aplicación.

Una vez hecho esto, Se deberá hacer click en “Run” y de este modo la aplicación se pondrá en marcha.