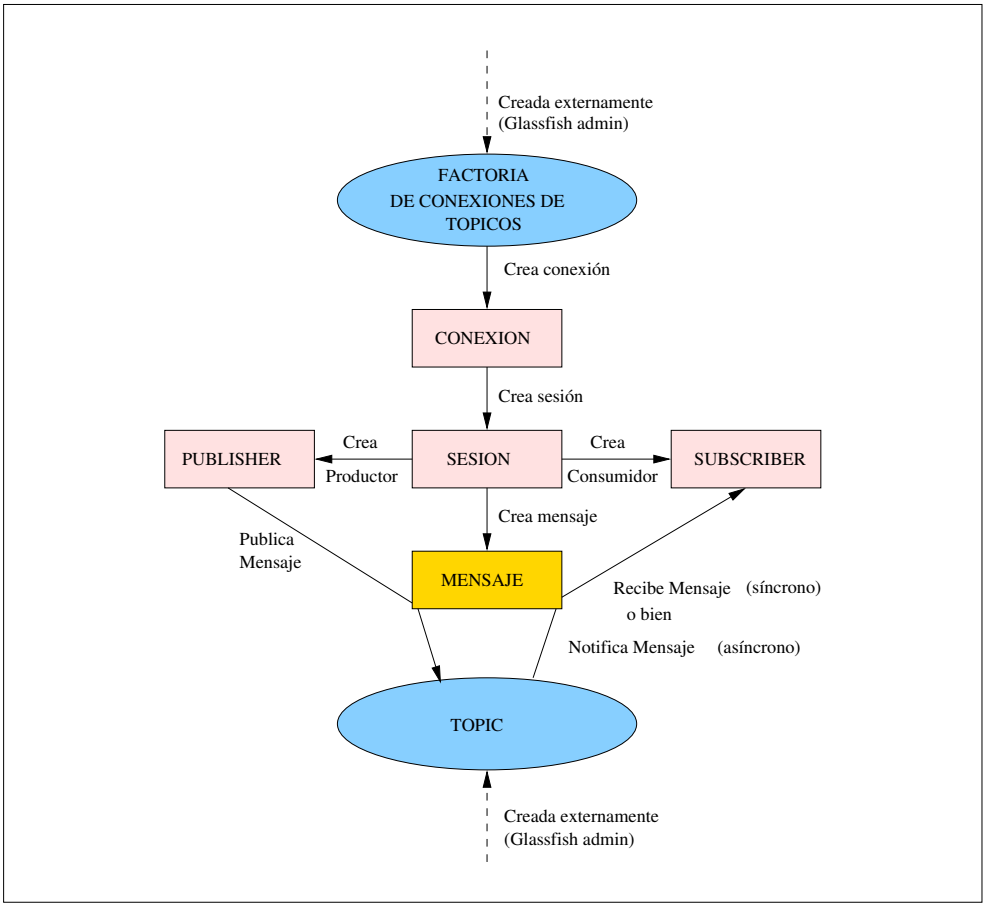
PRÁCTICAS DE SISTEMAS DISTRIBUIDOS.

Práctica 2: Publish/Subscribe con Enterprise Java Beans (EJB) y Java Message Service (JMS).

PRÁCTICA ENTREGABLE

En esta práctica mantendremos la división en grupos realizada al inicio de curso y trabajaremos con el paradigma *Publish/Subscribe* en el marco de la programación EJB. Uno o varios productores de mensajes (*Publishers*) publican eventos (mensajes), que los receptores o consumidores (*Subscribers*) reciben de manera asíncrona, mediante notificaciones. El soporte JMS de este paradigma *Publish/Subscribe* se basa en los denominados *topics*, similares a las colas de mensajes, a los que se pueden suscribir los clientes y recibir los mensajes destinados a un tópico en concreto.

La gráfica siguiente ilustra esquemáticamente el funcionamiento del modelo *Publish/Subscribe* en JMS:



Observamos que hemos de crear manualmente (mediante la herramienta de administración de recursos de *Glassfish*) una factoría de conexiones de tópicos y un *topic* (de manera similar a como se crean las colas de mensajes). Ambos recursos JMS se localizarán de nuevo mediante JNDI por parte de los potenciales clientes (*publishers/subscribers*).

A través de la factoría de conexiones de tópicos crearemos una conexión, la cual a su vez nos permitirá crear una sesión, de idéntica forma a lo realizado en la práctica 1. Una vez creada una sesión para un cliente (o servidor), a partir de ella podremos crear uno o varios productores/subscriptores de mensajes, los cuales ya pueden (respectivamente) publicar y recibir mensajes del *topic*, de forma síncrona o asíncrona. La recepción síncrona funciona de manera similar a lo realizado en la práctica 1, de modo que en esta práctica nos centraremos en la recepción asíncrona, mediante notificación, la cual se recibe cuando un productor publica un nuevo mensaje en el *topic*.

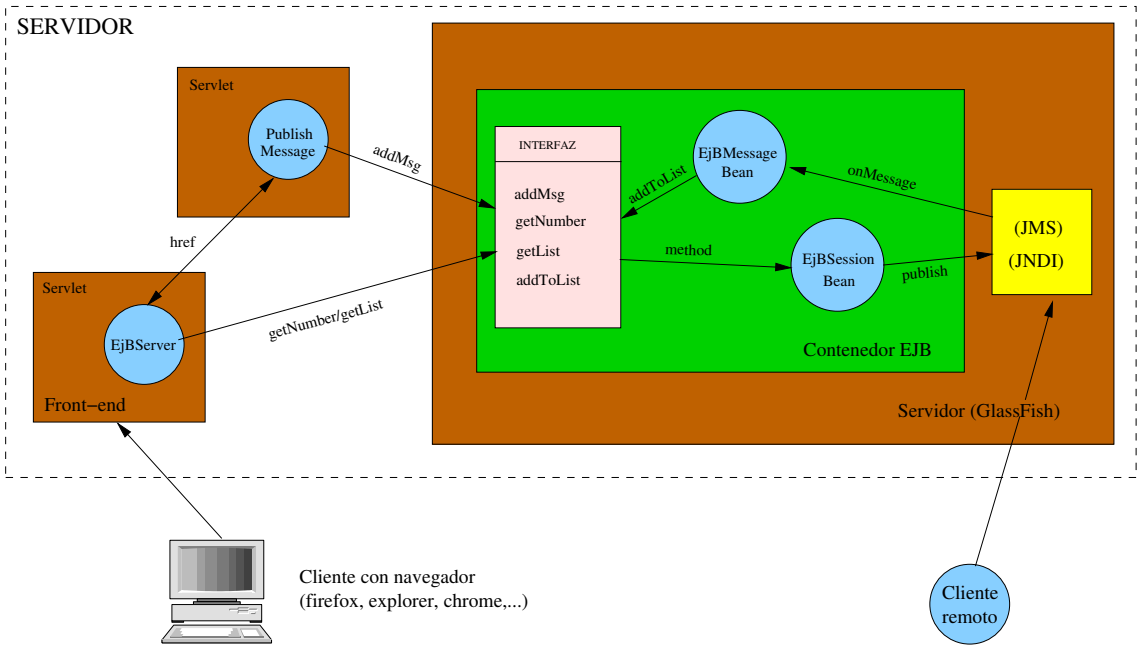
Por otra parte, Enterprise Java Beans (EJB) es una tecnología de componentes Java distribuidos, que permite implementar aplicaciones distribuidas de forma muy rápida, liberando al programador de muchos de los aspectos relacionados con la gestión de conectividad, recursos (JMS/JDBC), control de acceso, control de concurrencia, balanceo de carga, etc. En esta práctica combinaremos EJB con JMS para implementar un servidor de noticias muy simple, que gestiona un histórico de noticias, las cuales pueden ser publicadas de manera directa, accediendo al *topic* correspondiente, o bien vía web, a través de un *servlet* (servidor web que permite gestionar contenidos dinámicos).

En la práctica implementaremos y utilizaremos los siguientes elementos estructurales:

1. Clientes remotos que pueden publicar de forma directa noticias o recibir de manera asíncrona las noticias que hayan sido publicadas.

2. Del lado del servidor tendremos los recursos JMS (factoría de conexiones de tópicos y tópico concreto), un módulo web como front-end, que constará de dos servlets (EjbServer y PublishMessage), un Singleton Session Bean (EjbSessionBean), que implementará la lógica de negocio del servidor (publicar mensajes y gestionar un histórico de los mismos) y un Message-Driven Bean (EjbMessageBean), que permitirá al servidor recibir las notificaciones de los mensajes publicados, tanto de los que se publican usando el módulo web como los que se publiquen directamente de forma remota.

La figura siguiente ilustra la arquitectura de esta aplicación.



Los elementos principales son:

- *EjBSessionBean* es un Singleton Session Bean, del cual sólo se instancia una copia, que es compartida por todos los clientes. Es un tipo de bean que mantiene su estado entre invocaciones, aunque no mantendrá su estado en caso de caída o reinicio del servidor. Nos aprovechamos de este hecho para implementar el bean, de modo que mantendrá una lista de los mensajes publicados y un contador de los mismos. El histórico de los mensajes se guardará en un fichero de texto *mensajes.txt*, que deberá ser creado en caso de no existir (primera ejecución). De esta forma, al inicializar el bean éste obtendrá el histórico de mensajes desde el fichero, lo cual nos permite mantener el servicio incluso si el servidor es reiniciado.
- *EjbMessageBean* es un Message-Driven Bean, que permite al servidor recibir todos los mensajes publicados (directamente por clientes remotos o vía *EjbSessionBean*), y darlos de alta en el histórico.
- *EjBServer* es el front-end de la aplicación, permite a los clientes conectarse con un navegador, les muestra los mensajes actuales y el número de ellos, y pide si desean publicar un nuevo mensaje.
- *PublishMessage* es un servlet que se ejecuta para publicar un mensaje. Se limita a pedir el texto del mensaje e invocar el método *addMsg* del contenedor EJB para publicarlo. Pueden publicarse varios mensajes, o volver al servlet *EjBServer*.

La implementación de dichos elementos se repartirá entre los miembros del grupo de prácticas, debiendo quedar clara dicha división de trabajo.

A continuación se describen los pasos a seguir para realizar la práctica:

1. Entrar en *netbeans* y lanzar el servidor *Glassfish*. Crear desde la herramienta de administración tanto la factoría de conexiones de tópicos, como el tópico, con nombres JNDI *jms/FactoriaConexiones* y *jms/Noticias*, respectivamente. El nombre físico del topico será *Noticias*, y su tipo deberá ser *javax.jms.Topic*. Para la factoría de conexiones su tipo deberá ser *javax.jms.TopicConnectionFactory*.

   Si surgen problemas al crear los recursos con GlassFish de nuevo se podrán crear en línea de comandos:

   ```
   cd c:\Program Files\GlassFish-4.1.1\bin
   asadmin
     create-jms-resource --restype javax.jms.TopicConnectionFactory \
        --description "FactoriaConexiones" jms/FactoriaConexiones
     create-jms-resource --restype javax.jms.Topic --property Name="Noticias" jms/Noticias
   ```

2. Crear un nuevo proyecto "ProyectoEJB", de tipo *Java EE, Enterprise application*. Dejar marcadas las opciones de crear EJBModule y Web Application Module, que se corresponden con el contenedor EJB y el servidor web, respectivamente. Al pulsar *Finish* se generan 3 proyectos, el raíz *ProyectoEJB*, y sus 2 proyectos subordinados, *ProyectoEJB-ejb* (EJBModule) y *ProyectoEJB-war* (módulo web).

3. Para crear los dos beans, nos vamos al proyecto *ProyectoEJB-ejb*, lo desplegamos, y con el botón derecho del ratón sobre él creamos (New) un *SessionBean*, llamado *EjBSessionBean*. En *package* introducimos un nombre de paquete, por ejemplo *ejb*. Se elige de tipo *Singleton* y *Local*. Observemos que *Local* significa que será invocado por un cliente local, lo cual es cierto, pues los clientes remotos entran con un navegador al módulo web (servlets), y desde estos se producen las invocaciones (locales).

4. A continuación, pulsando de nuevo el botón derecho del ratón sobre el mismo proyecto *ProyectoEJB-ejb* creamos el bean *EjbMessageBean*, de tipo *Message-Driven Bean*. El nombre del paquete debe ser el mismo de antes (ejb), y debemos marcar *Server Destinations*, tomando el topic *jms/Noticias*. Pulsamos *next*, y nos muestra las propiedades del topic que va a utilizar. Por defecto la subscripción aparece como *DURABLE*, debe cambiarse a *NON_DURABLE*. Las subscripciones durables requieren un solo subscriptor y nosotros permitiremos varias subscripciones al tópico. Al finalizar nos muestra la clase con su método *onMessage*, que tendremos que completar posteriormente.

5. Para preparar los servlets desplegamos el proyecto *ProyectoEJB-war* y con el botón derecho del ratón (New) creamos un Servlet, de nombre *EjBServer*. Elegimos un nombre de paquete, por ejemplo ejb y pulsamos *Finish*. Nos crea un esqueleto de código para el método *processRequest*, que más tarde modificaremos.

6. De idéntica forma creamos un segundo *Servlet* de nombre *PublishMessage*, sobre el mismo paquete ejb.

7. Ya tenemos todos los elementos estructurales del lado del servidor, a falta de completar su código. **A partir de aquí los alumnos del grupo irán contribuyendo a la implementación según su responsabilidad**.

   Comenzaremos con el interfaz de *EjBSessionBean*, abrimos el código de *EjBSessionBeanLocal*, e incluimos los prototipos de los métodos:

```
void addMsg(String m);          // Publish New Message
int getNumber();                // Get number of Messages
LinkedList<String> getList();   // Get current list of news
void addToList(String m);       // Add a message to the list.
```

8. En *EjbSessionBean* pulsamos sobre el aviso a la izquierda para que incluya el código de los métodos abstractos que debe implementar. Eliminamos el código generado automáticamente en todos ellos para lanzar una excepción (Not supported yet). A continuación declaramos un contador entero y privado de mensajes a nivel de clase, *count_msg*, así como la lista de mensajes *list_msg*, los cuales deben inicializarse.

   EjBSessionBean debe publicar mensajes (método *addMsg*), de modo que tiene que acceder a la factoría de conexiones de tópicos y al tópico en concreto que hemos creado. Hay dos formas de hacer esto, la primera es similar a lo realizado en la práctica 1, obteniendo el contexto JNDI actual y localizando directamente en él ambos recursos:

```
InitialContext iniCtx = new InitialContext();
Object tmp = iniCtx.lookup("jms/FactoriaConexiones");
TopicConnectionFactory tcf = (TopicConnectionFactory) tmp;
Topic t = (Topic) iniCtx.lookup("jms/Noticias");
```

La segunda opción consiste en usar la inyección de código, mediante anotaciones Java a nivel de clase (justo inmediatamente después de la declaración de la clase):

```
@Resource(mappedName="jms/FactoriaConexiones")
private TopicConnectionFactory tcf;

@Resource(mappedName="jms/Noticias")
private Topic t;
```

- El método *addMsg* deberá crear en ambos casos la conexión (*TopicConnection*), usando el método *createTopicConnection* sobre la factoría de conexiones, la sesión (método *createTopicSession* sobre el objeto *TopicConnection*) y el *publisher* (método *createPublisher* sobre la sesión). Una vez creado el publisher, éste debe iniciarse (método *start* de *TopicConnection*). Con todo ello ya podrá publicar el mensaje (método *publish* del *publisher*), y cerrar el publisher (método *close*). La conversión del mensaje al tipo requerido se hará como en la práctica 1.
- El método *getNumber* se limita a devolver el valor del contador, de igual forma que *getList* devuelve la lista actual. El método *addToList* añade un mensaje a la lista e incrementa el contador.
- *EjBSessionBean* necesita un código de inicialización, para cargar el histórico de mensajes desde el fichero *mensajes.txt* en la lista y asignar el valor del contador al número de mensajes sobre el fichero (cada línea de texto es un mensaje). Si el fichero no existe, éste será creado en esta inicialización.

  Para implementar esta inicialización usaremos la anotación EJB "*@PostConstruct*" sobre un método *inicializacion*, que de esta forma será ejecutado al iniciarse el bean.

  ```
  @PostConstruct
  void inicializacion(){
    < escribir codigo de inicializacion>
  }
  ```

  El bean se ejecuta por GlassFish, los mensajes que quieras mostrar por el bean debes buscarlos en el log del servidor GlassFish.

9. Pasamos ahora a completar el código de *EjbMessageBean*, del cual básicamente sólo hemos de completar el método *onMessage*. Este bean necesita comunicarse con *EjBSessionBean*, de modo que a nivel de la clase con el botón derecho del ratón pulsamos *InsertCode, Call Enterprise Bean*. Seleccionamos el proyecto *ProyectoEJB-ejb* y lo desplegamos, tomando *EjBSessionBean*. Con ello se realiza una nueva inyección de código EJB para permitirnos el acceso al bean. El código a completar en el método *onMessage* debe abrir el fichero *mensajes.txt* en modo *append*, escribir sobre él el mensaje recibido e invocar el método *addToList* de *EjBSessionBean* para darlo de alta en la lista que éste almacena.

10. Nos movemos ahora al módulo web, vamos a preparar el interfaz para los clientes a través del servlet *EjBServer*. Procedemos como en el apartado anterior para inyectar código que nos permita invocar a *EjBSessionBean*. Una vez hecho esto modificamos el código HTML generado por el servlet, para incluir un enlace al servlet *PublishMessage*, así como para mostrar las noticias actuales, y el número de ellas. Puedes modificar a tu gusto el título de la página web, y la primera cabecera, e inmediatamente detrás incluir el enlace al servlet *PublishMessage*:

```
out.println("<a href='PublishMessage'> Publicar noticia</a>");
out.println("<br><br>");   // 2 saltos de linea
```

Justo debajo puedes incluir el histórico de noticias, obtenido a partir de *getList*, así como el número total de ellas (*getNumber*). Para mostrarlas debes seguir el mismo formato que el indicado: *out.println(...)*.

11. Para completar el código de *PublishMessage* procedemos de nuevo a inyectar código para acceder al session bean. Una vez hecho eso modificamos el código HTML generado automáticamente. Incluimos el código siguiente tras la asignación del tipo de codificación de caracteres (UTF-8) para recoger el valor tecleado como mensaje:

```
String m=request.getParameter("MESSAGE");
if (m != null){
   ejBSessionBean.addMsg(m);
}
```

Puedes modificar el título y primera cabecera a tu gusto, a continuación creamos un formulario HTML para introducir el texto del mensaje y un enlace de retorno a la página anterior.

En *http://www.w3schools.com/html/html_forms.asp* puedes ver los elementos que puedes incluir para hacerlo más presentable, nosotros nos limitamos a incluir una entrada de texto:

```
out.println("<form>");
out.println("NOTICIA:  <input type='text' name='MESSAGE'><br/>");
out.println("<input type='submit' value='Submit'><br/>");
out.println("<br><br><br>");
out.println("<a href='EjBServer'> Volver a pagina anterior</a>");
out.println("</form>");
```

12. Ya podemos lanzar el servidor, aunque para ello debes entrar en *Properties* del proyecto raíz (botón derecho del ratón, abajo del todo), y seleccionar *Run*. En *Relative URL* incluir *EjBServer*, para que ejecute el servlet al conectarse los clientes. A continuación puedes hacer el despliegue (desde proyecto raíz, *deploy*), y comprobar si todo ha ido bien en los mensajes de error de *GlassFish*. Si el despligue ha sido correcto puedes ejecutarlo para probarlo (*Run* desde raíz). Prueba también a conectarte de forma remota desde otro ordenador con un navegador, introduciendo la URL:

```
http://IP_MAQUINA:8080/ProyectoEJB-war/EjBServer
```

13. Finalmente, debe crearse un cliente remoto, que se conectará al *topic NOTICIAS* de forma remota (tomar como referencia la práctica 1 para ello) y permitirá dar de alta nuevas noticias, a la vez que se subscribirá para recibir las noticias publicadas, las cuales mostrará en pantalla.

Las clases a utilizar en este caso son *TopicConnectionFactory, TopicConnection, TopicSession, TopicPublisher* y *TopicSubscriber*. El *Publisher* se crea a partir de un objeto de la clase *TopicSession*. El *publisher* también actúa como subscriptor, de modo que se subscribe con *createSubscriber* (*TopicSession*). Desde el objeto de la clase *TopicSubscriber* obtenido ejecutará el método *setMessageListener* sobre un nuevo objeto de la clase *MsgListener* que le

permitirá quedar a la escucha de los mensajes entrantes. Habrá que implementar el método *onMessage* de dicha clase para mostrar los mensajes que van siendo publicados.

Por tanto, las recepciones son **asíncronas**, se muestran cuando llegan, mientras tanto, el cliente estará pidiendo indefinidamente mensajes por teclado para publicarlos.

14. Esta práctica es entregable, mediante fichero ZIP con proyectos **exportados** desde Netbeans y memoria en PDF documentando el trabajo realizado, que incluya comentarios al código y capturas de pantalla mostrando su correcto funcionamiento. Es suficiente con que uno de los miembros del equipo realice la entrega.

15. Prueba en laboratorio ante el profesor. La prueba se puede hacer al acabar la práctica (preferible) o bien en la evaluación presencial final de prácticas. Probaremos la práctica en un entorno distribuido, con **al menos dos clientes conectándose vía web y dos clientes que acceden al tópico de forma remota, sin utilizar el servidor web**. Recuerda desactivar el cortafuegos para probar el cliente remoto.