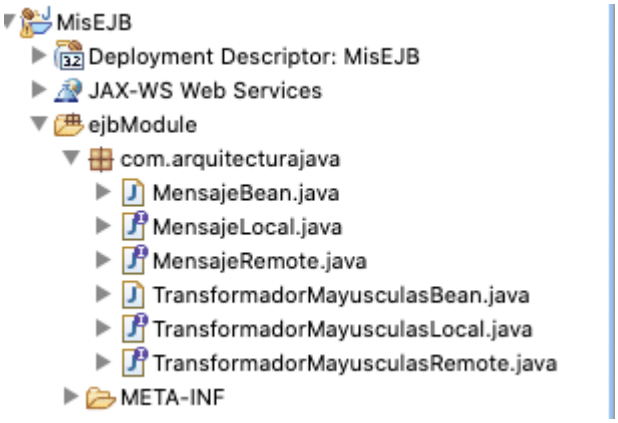
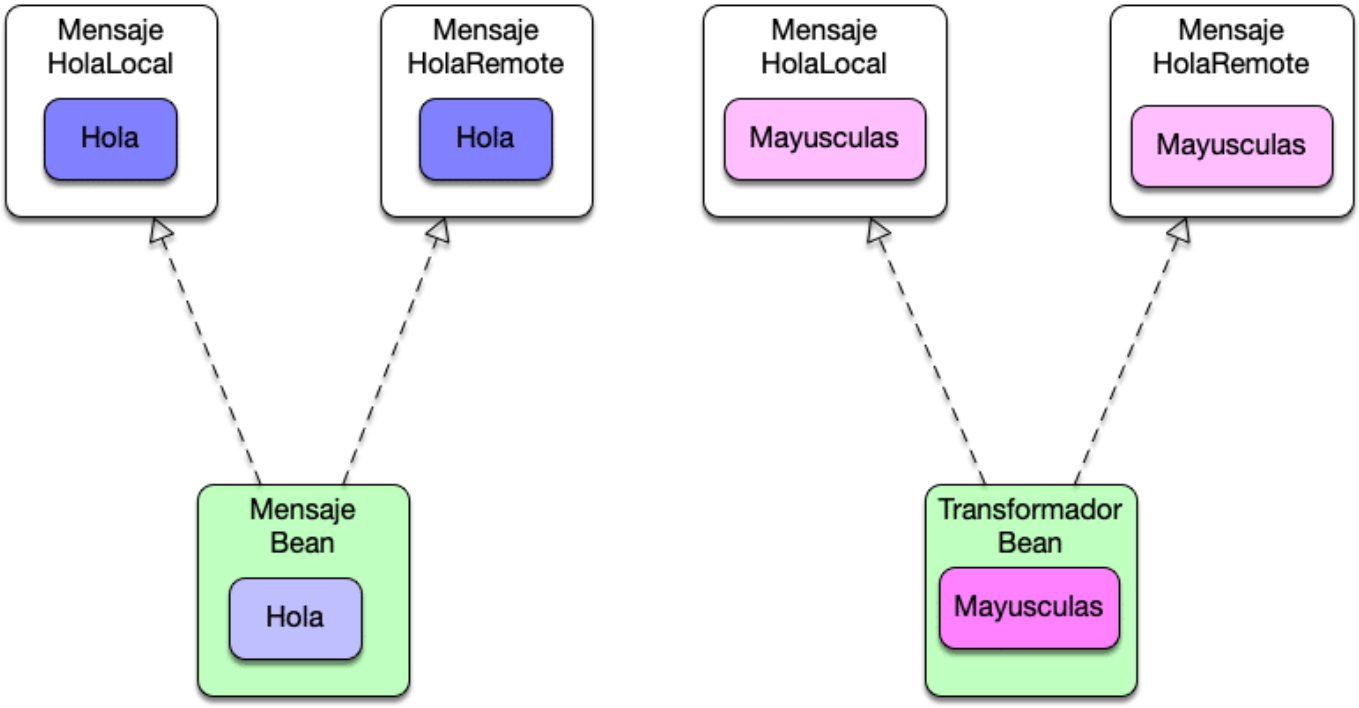
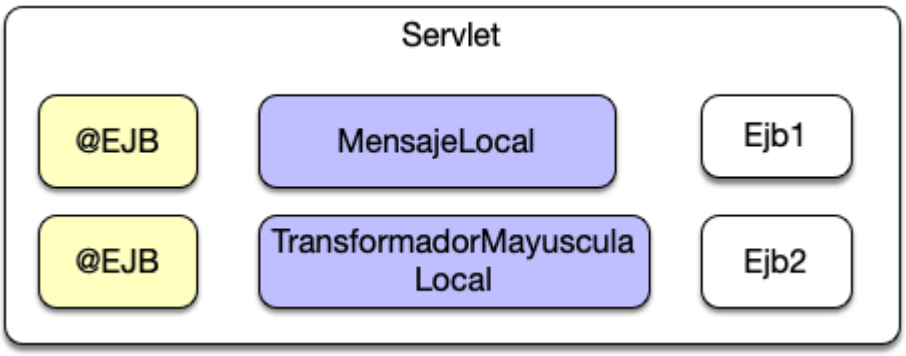
Cuando trabajamos con EJBs estamos más que acostumbrados a construir un montón de EJBs estos EJBs normalmente disponen de un interface que define la funcionalidad y un Bean que la implementa . Esto en principio lo hace todo relativamente sencillo . Sin embargo existen situaciones en las que podemos necesitar usar de forma independiente los interfaces y empaquetarlos en un JAR esto es lo que habitualmente se conoce como EJB Client. Vamos a ver cómo podemos construir este tema con Eclipse. Para ello vamos a partir de un proyecto de EJBs que dispone de dos EJBS previamente construidos que son muy muy sencillos.



Pueden no parecerlos pero se trata de dos EJBS que implementan interfaces Locales y Remotos. El primer EJB es MensajeBean que contiene un método de hola y el segundo EJB es TransformadorMayusculas que contiene un método mayúsculas que nos convierte un texto a mayúsculas.



Nosotros podemos usar este proyecto en un proyecto Web y construir un servlet que acceda a cada uno de los EJBS para imprimir en una pagina web el resultado de la transformación y en la consola el Hola.



Vamos a ver el código tanto de los EJBs como del Servlet

```
@Local
public interface TransformadorMayusculasLocal {

        public String mayusculas(String texto);
}
```

```
package com.arquitecturajava;

import javax.ejb.Remote;

@Remote
public interface TransformadorMayusculasRemote {

	public String mayusculas(String texto);
}
```

```
package com.arquitecturajava;

import javax.ejb.Stateless;

/**
 * Session Bean implementation class TransformadorMayusculas
 */
@Stateless
public class TransformadorMayusculasBean implements
TransformadorMayusculasRemote, TransformadorMayusculasLocal {


	@Override
	public String mayusculas(String texto) {
		return texto.toUpperCase();
	}

}
```

Una vez que tenemos el código del EJB es momento de mostrar el código del Servlet.

```
package com.arquitecturajava.web;
```

```
import java.io.IOException;

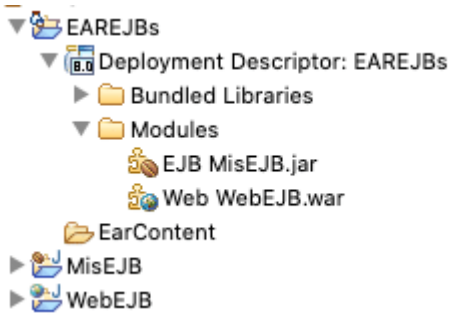
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.arquitecturajava.MensajeLocal;
import com.arquitecturajava.TransformadorMayusculasLocal;

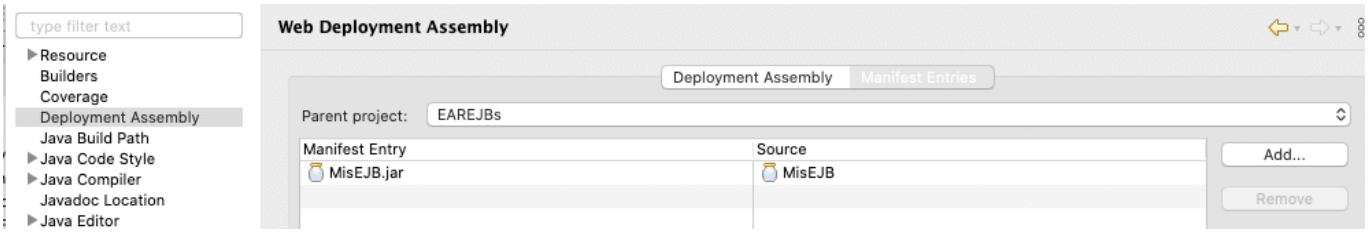
/**
 * Servlet implementation class ServletEJB
 */
@WebServlet("/ServletEJB")
public class ServletEJB extends HttpServlet {
        private static final long serialVersionUID = 1L;
        @EJB
        MensajeLocal ejb1;
        @EJB
        TransformadorMayusculasLocal ejb2;
        protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                ejb1.hola();
                response.getWriter().append(ejb2.mayusculas("hola que
tal"));
        }
```

}

Una vez que tenemos todo configurado nos tenemos que dar cuenta que el Servlet accede a los EJBs a través de sus interfaces locales. Esto lo puede hacer relativamente sencillo ya que los he incluido en Eclipse en un proyecto EAR (Enterprise Application Project.



Hecho esto ademas los he referenciado a través del menu de deployment Assembly.



Esto nos permite ejecutar los EJBs via el Servlet y obtener los resultados tanto en la consola como en la web.

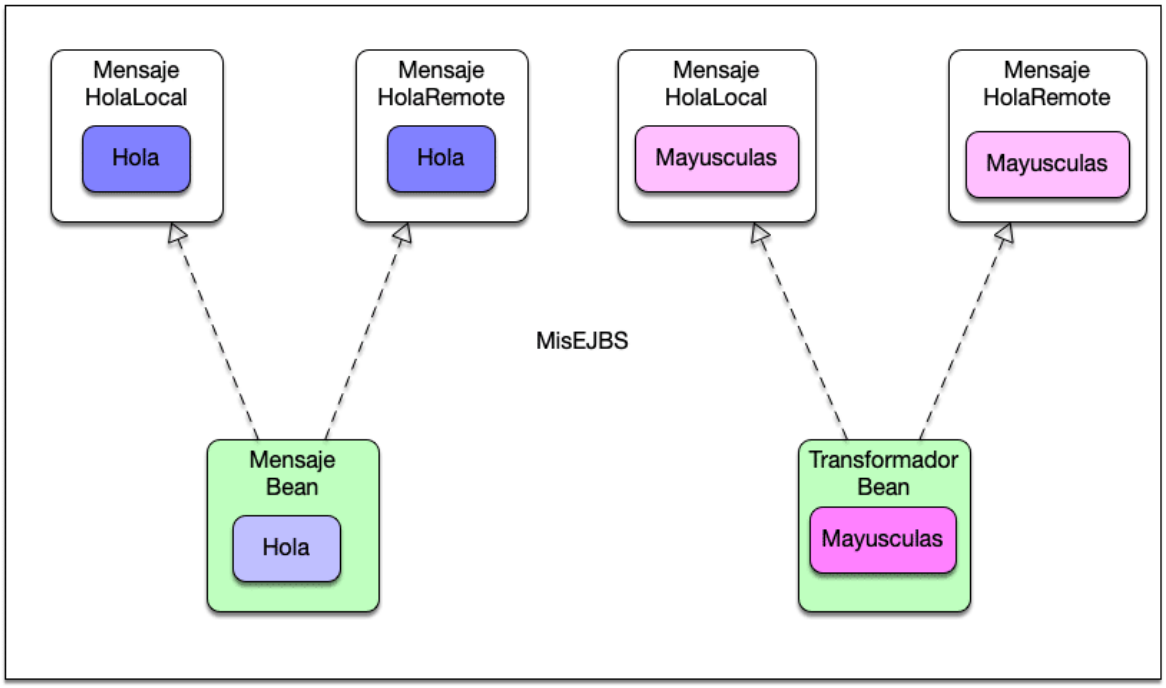



```
HOLA QUE TAL
```

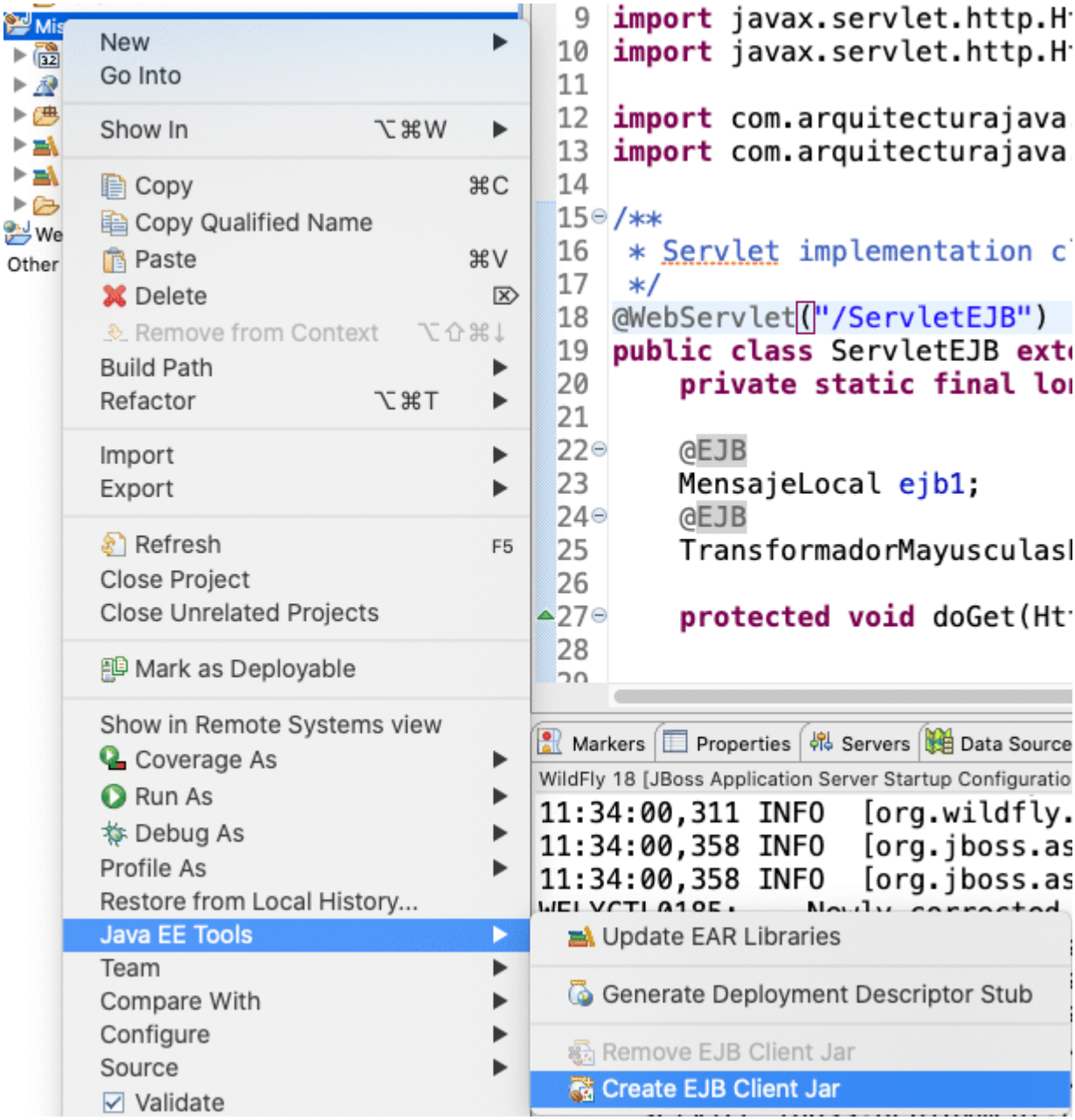
EJBClient y extensibilidad

¿Ahora bien es esto correcto?. Parece que si porque nos es suficiente para ejecutarlo todo . Sin embargo existe un problema importante de extensibilidad . Si nosotros desde otro proyecto por ejemplo un proyecto de consola queremos referenciar a los EJBs . No nos quedará más remedio que incluir el JAR de “MisEJBS” el cual incluye los interfaces y la

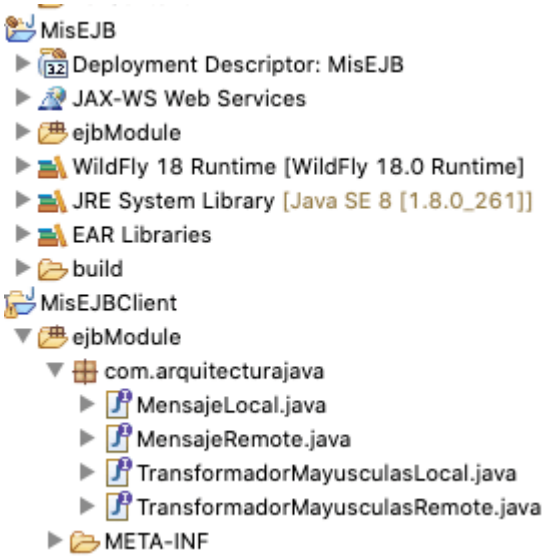
implementación de las clases .



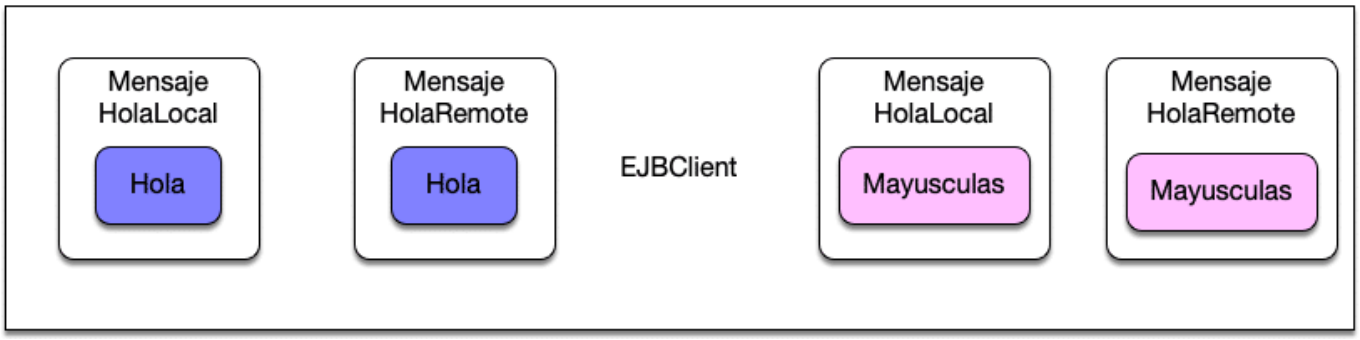
Esta más que claro que una aplicación de consola no necesita para nada la implementación de los EJBS … simplemente necesita los interfaces para invocarlos remotamente. ¿Cómo podemos hacer el código más flexible? . Muy sencillo podemos pedirle a Eclipse que nos genere un EJBClient



Esto transformará nuestro proyecto de EJBS y publicará un nuevo proyecto de Eclipse que incluya únicamente los interfaces. Para que otras aplicaciones lo puedan usar con mayor flexibilidad.



Nos hemos quedado en el EJB Client únicamente con los interfaces.



EJB Client Simplificación

Si desplegamos el proyecto original de EJBs este se habrá quedado únicamente con los beans esenciales y no con las implementaciones .Eclipse se encarga de la refactorización de forma transparente.



Otros artículos relacionados

- Usando un EJB Async
- El concepto de EJB in WAR y su uso
- EJB Singleton
- Curso Java Web