

Taller de Sistemas de Información 2

La plataforma Java EE

19 de Agosto de 2014



Instituto de
Computación



Facultad de
Ingeniería



Universidad de la
República de Uruguay

Visión general

- ❑ Las organizaciones existen hoy día en un ambiente altamente competitivo
- ❑ Se necesitan aplicaciones que permitan cumplir sus requisitos de negocio, las cuales son cada vez mas complejas

- ❑ Vivimos en un mundo globalizado
 - Las compañías se encuentran distribuidas en diferentes continentes
 - Hacen negocios 24/7, a través de internet y en diferentes países
 - Tienen que lidiar con múltiples datacenters, en diferentes zonas horarias y en diferentes culturas
- ❑ Todo esto, a la vez que se busca reducir costos, mejorar los servicios y aumentar la confiabilidad

- ❑ Además de esto, las compañías tienen que combinar todo esto con los sistemas existentes
- ❑ Y esto debe hacerse sin perder dinero, minimizando (o eliminando) los tiempos de caídas, haciendo que los sistemas estén disponibles 24/7, sean seguros y a su vez escalables

- ❑ En resumen, las aplicaciones empresariales deben enfrentarse con el cambio y la complejidad, deben ser flexibles y deben ser robustas
- ❑ Encarar este tipo de tareas sin ayuda, es realmente complejo
- ❑ **Este es el motivo por el cual fue creada la plataforma Java EE**

- ❑ Las primeras versiones de Java EE (antes conocida como J2EE) se focalizaban en resolver los problemas de las empresas en 1999
 - El problema de los componentes distribuidos
 - Como lograr distribución y que el costo de desarrollar y mantener estos no fuese prohibitivo
- ❑ Desde entonces las aplicaciones se han tenido que adaptar a nuevas tecnologías
 - Por ejemplo, los web services SOAP o REST

- ❑ La plataforma ha evolucionado a lo largo del tiempo para responder a estos avances (cambios) tecnológicos
 - Siempre a través del uso de estándares
- ❑ Con el paso del tiempo, Java EE ha cambiado, volviéndose una plataforma mas rica, mas simple, mas fácil de usar, mas portable y mas integrada

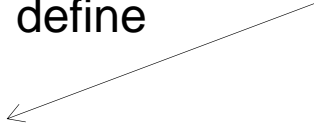
- ❑ Cuando queremos trabajar con colecciones en Java, no empezamos desarrollando nuestra propia Hashtable
 - **Usamos el API de colecciones (java.util.*) !**
 - (o por lo menos deberíamos)
- ❑ Lo mismo sucede si queremos desarrollar una aplicación web que funcione en un ambiente transaccional
 - Usamos la versión empresarial de Java

- ❑ Java EE provee una forma estándar para:
 - Manejar transacciones con el API JTA
 - Manejar mensajería con el API JMS
 - Manejar persistencia con el API JPA
- ❑ Es un conjunto de especificaciones orientadas al desarrollo de aplicaciones empresariales
- ❑ Puede ser vista como una extensión de la plataforma Java

Especificación Java EE

Especificación Java EE

define



Reglas p/Aplicaciones

Especificación Java EE

```
graph TD; A[Especificación Java EE] -- define --> B[Reglas p/Aplicaciones]; A -- define --> C[Reglas p/Sw. Base];
```

define

Reglas p/Aplicaciones

define

Reglas p/Sw. Base

Especificación Java EE

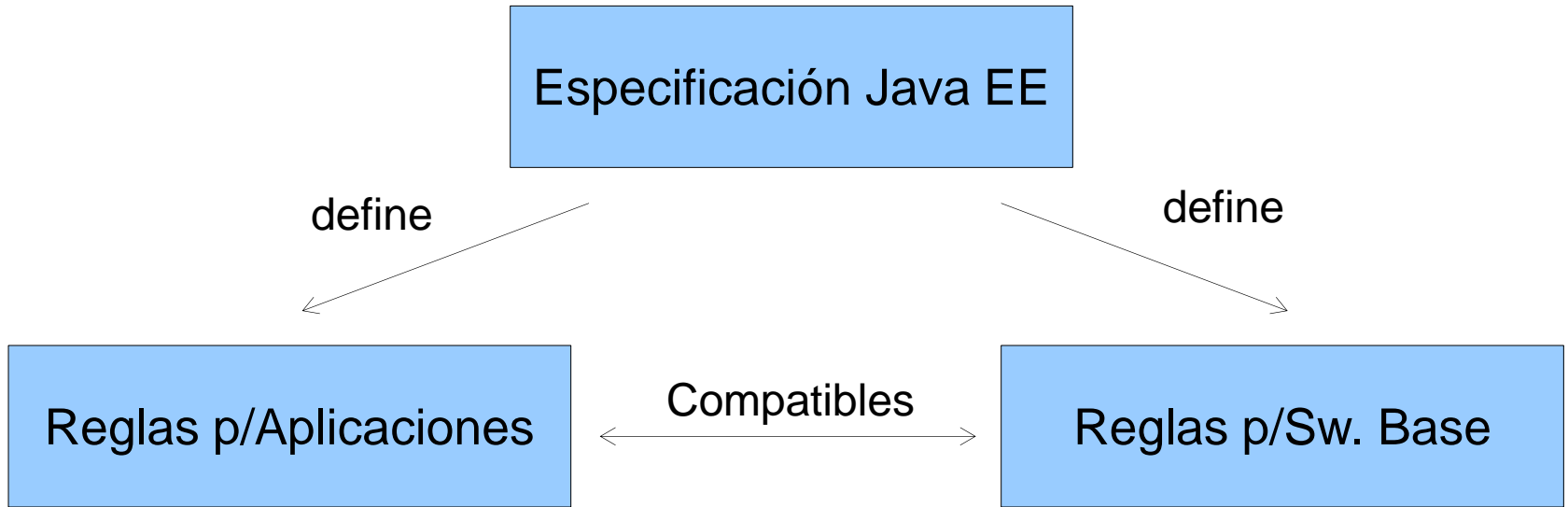
define

define

Reglas p/Aplicaciones

Compatibles

Reglas p/Sw. Base



Especificación Java EE

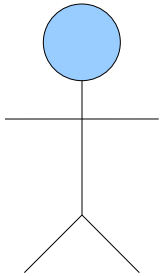
define

define

Reglas p/Aplicaciones

Compatibles

Reglas p/Sw. Base



Desarrollador
Aplicaciones

Especificación Java EE

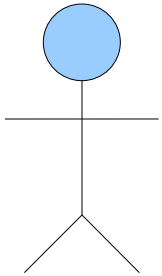
define

define

Reglas p/Aplicaciones

Compatibles

Reglas p/Sw. Base



Desarrollador
Aplicaciones

construye

Aplicación Empresarial

Especificación Java EE

define

define

Reglas p/Aplicaciones

Compatibles

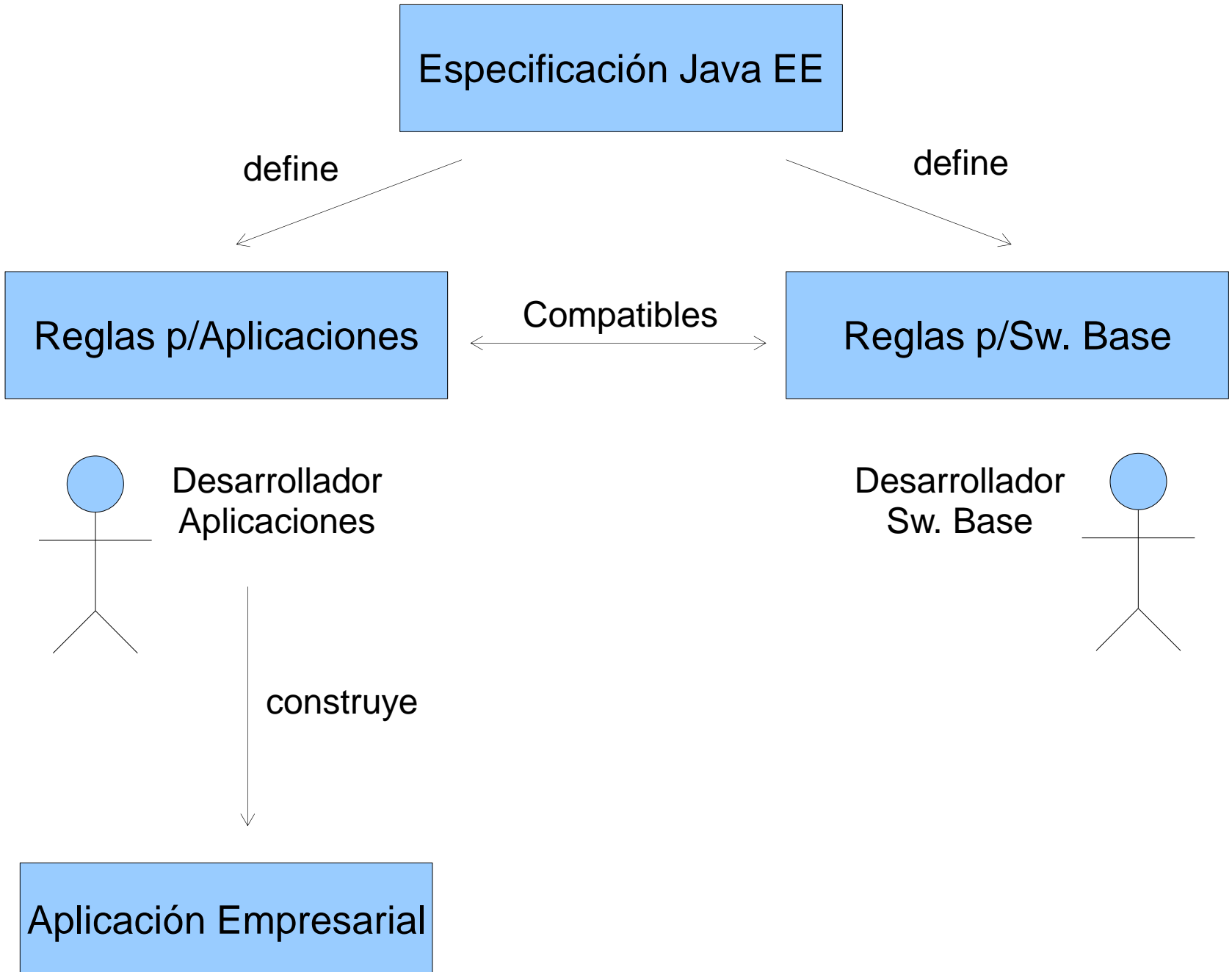
Reglas p/Sw. Base

Desarrollador
Aplicaciones

Desarrollador
Sw. Base

construye

Aplicación Empresarial



Especificación Java EE

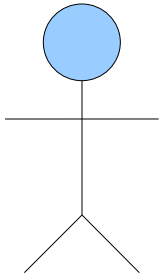
define

define

Reglas p/Aplicaciones

Compatibles

Reglas p/Sw. Base

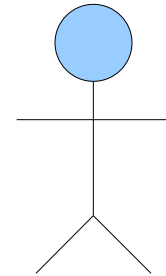


Desarrollador
Aplicaciones

construye

Aplicación Empresarial

Desarrollador
Sw. Base



construye

Servidor de Aplicaciones

Especificación Java EE

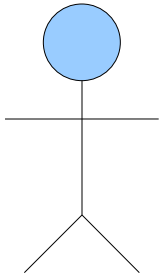
define

define

Reglas p/Aplicaciones

Compatibles

Reglas p/Sw. Base

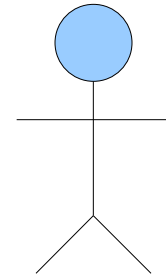


Desarrollador
Aplicaciones

construye

Aplicación Empresarial

Desarrollador
Sw. Base



construye

Servidor de Aplicaciones

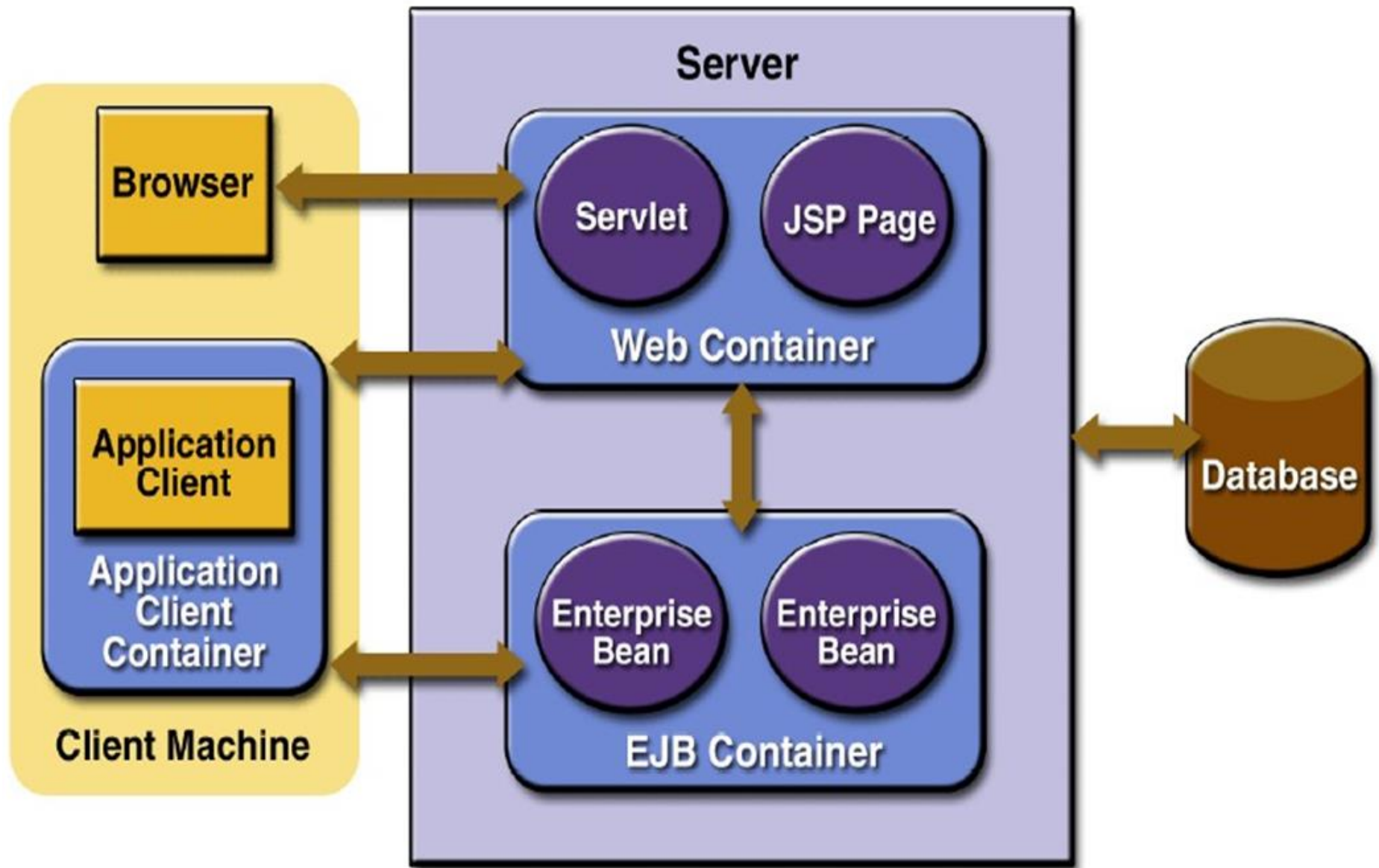
Compatibles

- ❑ Es una versión muy importante de Java EE
- ❑ Sigue los pasos de Java EE simplificando el modelo de desarrollo de aplicaciones
- ❑ Pero además agrega nuevas especificaciones (muy importantes) y nuevas funcionalidades a especificaciones existentes
- ❑ Hoy día es una plataforma muy bien documentada, con una comunidad muy grande y muy experimentada

- ❑ Java EE es una especificación implementada por una serie de contenedores
- ❑ Un contenedor es un runtime para las aplicaciones Java EE
 - Provee servicios para los componentes allí instalados
 - Manejo del ciclo de vida, inyección de dependencias, acceso a recursos, concurrencia, etc.
- ❑ Se agrupan en una entidad denominada “Servidor de aplicaciones”

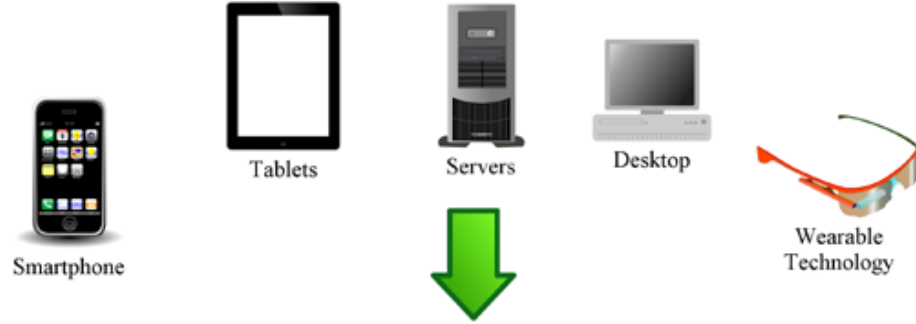
Servidor de aplicaciones

- ❑ Es un software que provee un ambiente de ejecución para las aplicaciones
- ❑ Le brindan a los componentes, la facilidad para poder acceder a los recursos necesarios para funcionar
- ❑ Es un host para los diferentes contenedores que utiliza la aplicación

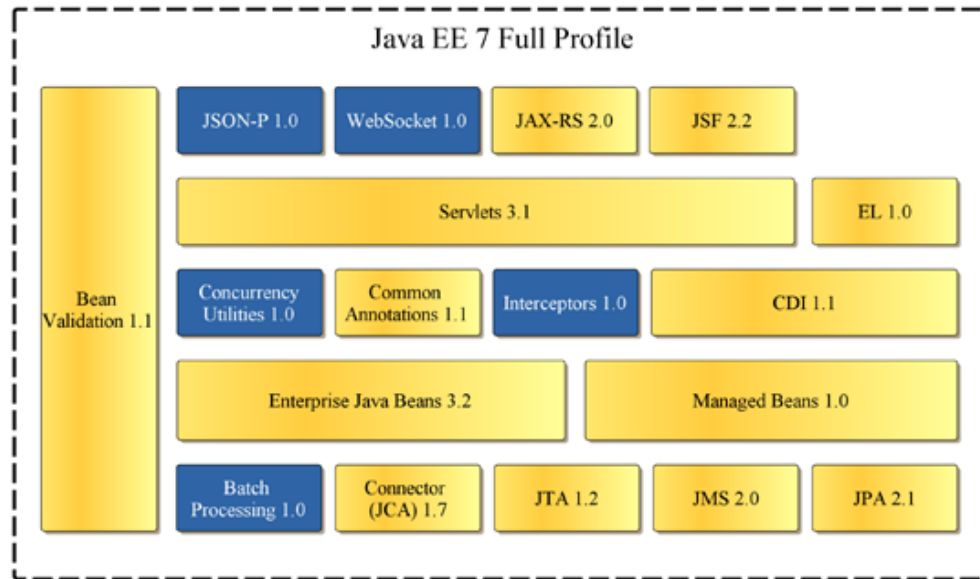


Servidor de aplicaciones

- ❑ Apache Tomcat
- ❑ Jboss / Wildfly
- ❑ Glassfish
- ❑ Websphere
- ❑ Oracle Application Server
- ❑ Jetty
- ❑ Caucho

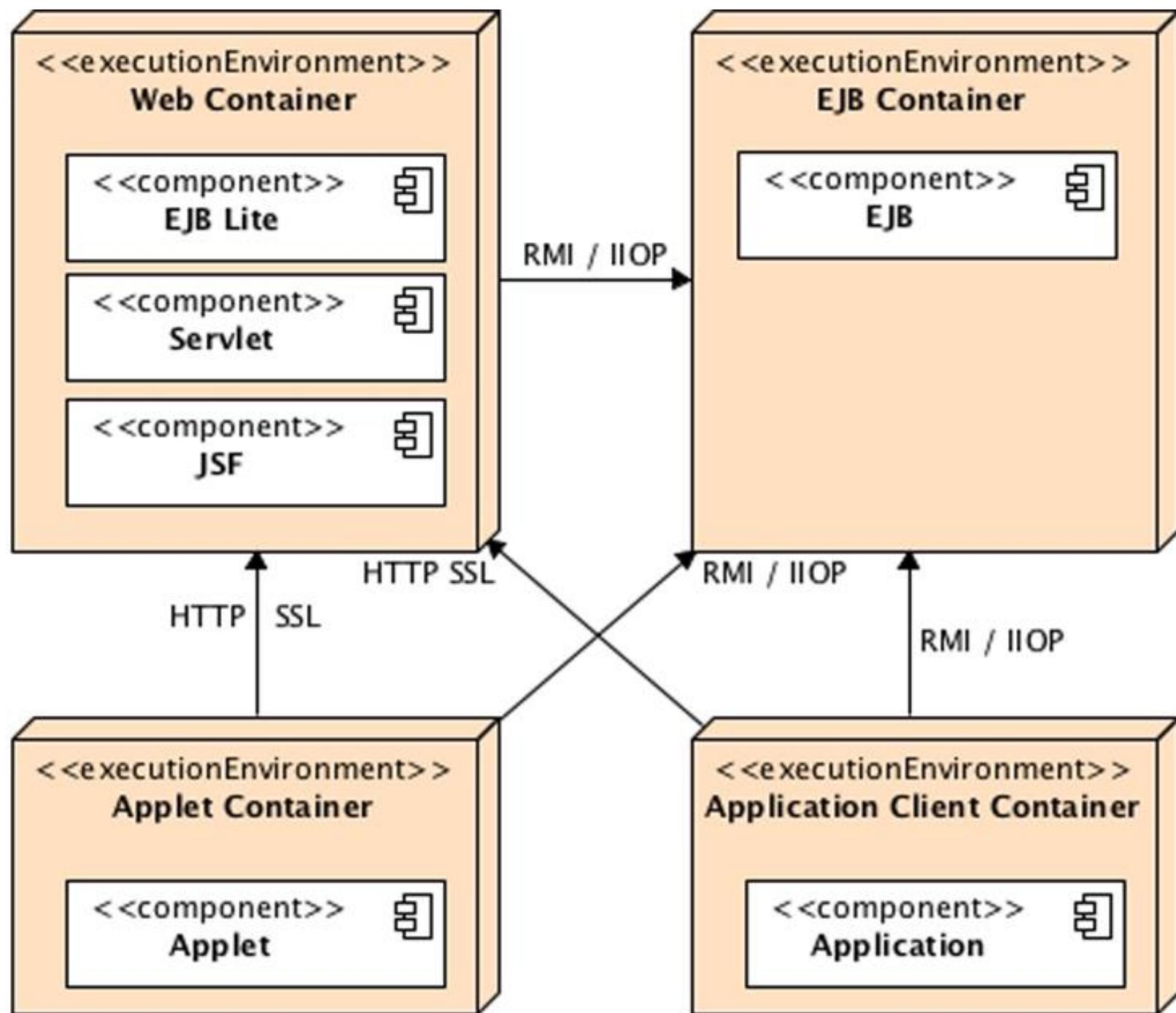


The Client Side



The Infrastructure Side





- ❑ Estos componentes utilizan una serie de contratos bien definidos para comunicarse con la infraestructura Java EE y con otros componentes
- ❑ Deben además ser empaquetados siguiendo una estructura de directorios predefinida (que luego veremos) antes de poder ser instalados

- ❑ La plataforma Java EE define cuatro tipos de módulos a debe soportar
 - **Aplicaciones empresariales:** Compuestas por EJBs, servicios JMS, componentes transacciones, etc. Son componentes orientados al procesamiento de lógica de negocio transaccional

- ❑ La plataforma Java EE define cuatro tipos de módulos a debe soportar
 - **Applets:** Son aplicaciones (GUI) que ejecutan dentro de un browser
 - **Aplicaciones cliente:** Son programas que ejecutan en el cliente, accediendo a los servicios del backend
 - **Aplicaciones web:** Ejecutan en un web container y responden a solicitudes HTTP desde un browser

- ❑ La infraestructura de Java EE esta particionada en contenedores
- ❑ Cada contenedor:
 - Tiene un rol especifico
 - Soporta una serie de APIs
 - Ofrece una serie de servicios
- ❑ Esconden complejidad técnica y aumentan la portabilidad

- ❑ Dependiendo del tipo de aplicación que vayamos a armar, deberemos entender las capacidades y limitaciones de los contenedores involucrados
- ❑ Por ejemplo, si armamos una aplicación web:
 - Si no necesitamos lógica transaccional remota, podemos utilizar un solo contenedor
 - Pero si tenemos que invocar lógica remota o utilizar mensajería, necesitaremos mas de un contenedor

❑ Applet Containers

- Provistos por el browser para ejecutar applets
- Este contenedor utiliza un modelo de sandbox para garantizar la seguridad del ambiente de ejecución

❑ Application Cliente Containers

- Consisten en las clases y bibliotecas necesarias para desarrollar aplicaciones que dialoguen con los contenedores Web y EJB

❑ Web Container

- Provee los servicios necesarios para ejecutar componentes web
- Servlets, JSP, Filtros, JSF, Listeners, Web Services y EJB lite
- Es responsable de soportar el protocolo HTTP y HTTPS entre el cliente y el servidor
- Las paginas web accedidas por el cliente, son generadas en este contenedor

❑ EJB Container

- Es el responsable por manejar la ejecución de los componentes EJB
- Session Beans y Message Driven Beans
- Contienen la lógica de negocio transaccional (potencialmente remota) de la aplicación
- Provee servicios como la gestión del ciclo de vida, transacciones, mensajería, seguridad, concurrencia, distribución, nombrado, etc.

- ❑ Los contenedores proveen servicios a los componentes instalados
- ❑ Como desarrollador, permiten que nos concentremos en el desarrollo de los aspectos relacionados con el negocio
- ❑ Los problemas de base relacionados a las aplicaciones empresariales, son resueltos por estos servicios

□ Java Transaction API

- Provee un servicio para el demarcado de transacciones, para ser usado desde las aplicaciones
- También provee una interfaz entre el gestor de la transacción y el gestor de recursos (que en ultima instancia provee la implementación de la transacción)

❑ Java Persistence API

- Es el API estándar para mapeos objeto – relacional (ORM)
- Brinda un lenguaje de consulta (JPQL) para que se pueda acceder en forma orientada a objetos a la información almacenada en la base

❑ Bean Validation

- Provee mecanismos para realizar validaciones a nivel de métodos y clases

- ❑ Java Message Service API
 - Permite que los componentes se comuniquen en forma asíncrona a través de mensajes
 - Soporta comunicación confiable usando un modelo P2P o Pub-Sub
- ❑ Java Naming and Directory Interface API
 - Es usada para acceder a servicios de nombrado y directorio
 - Permite asociar nombres a objetos, así como localizar los mismos a través de sus nombres

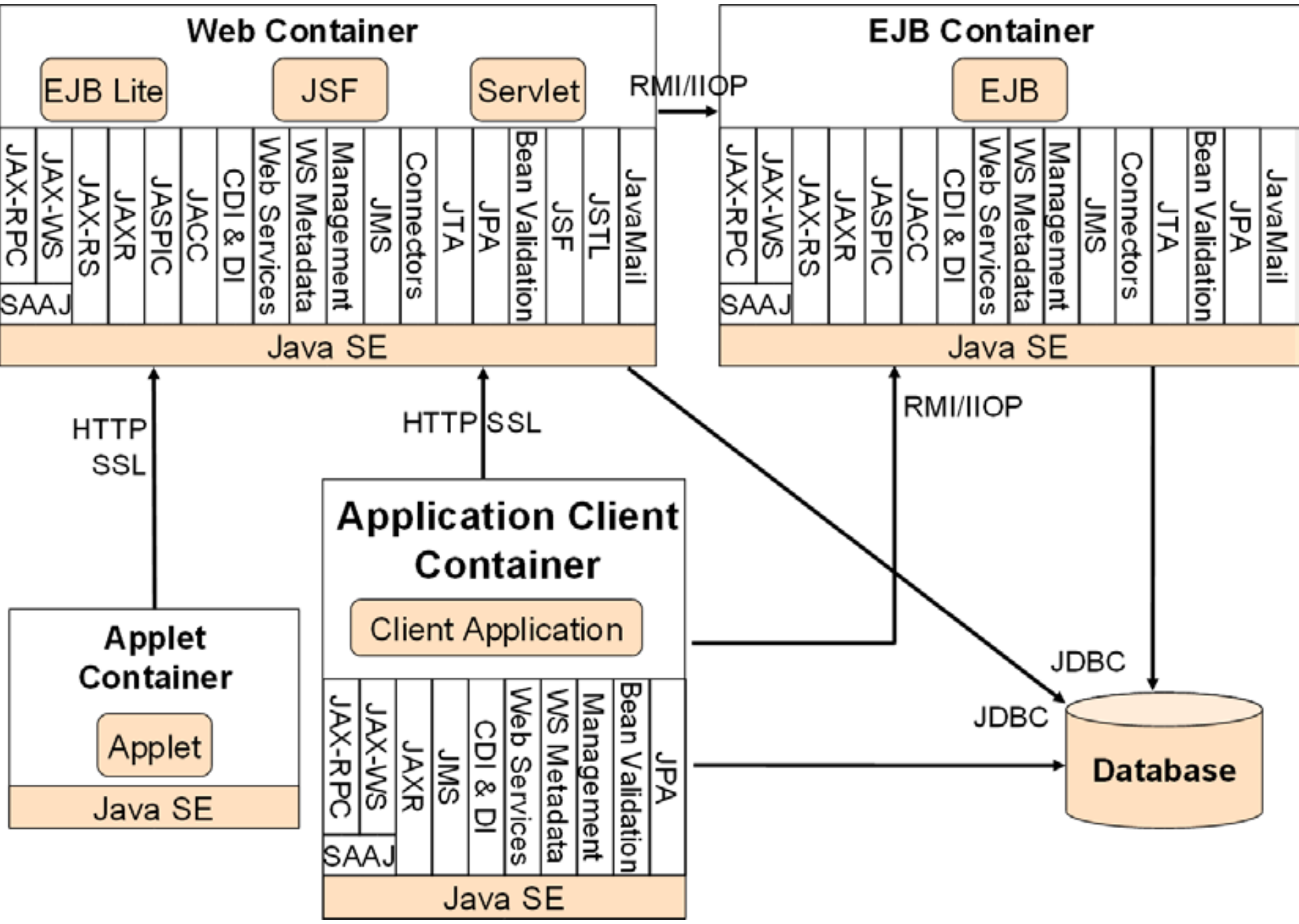
❑ Seguridad

- Java Authentication and Authorization Service (JAAS) provee los mecanismos para resolver la autenticación y autorización necesarias para acceder a los diferentes componentes de una aplicación

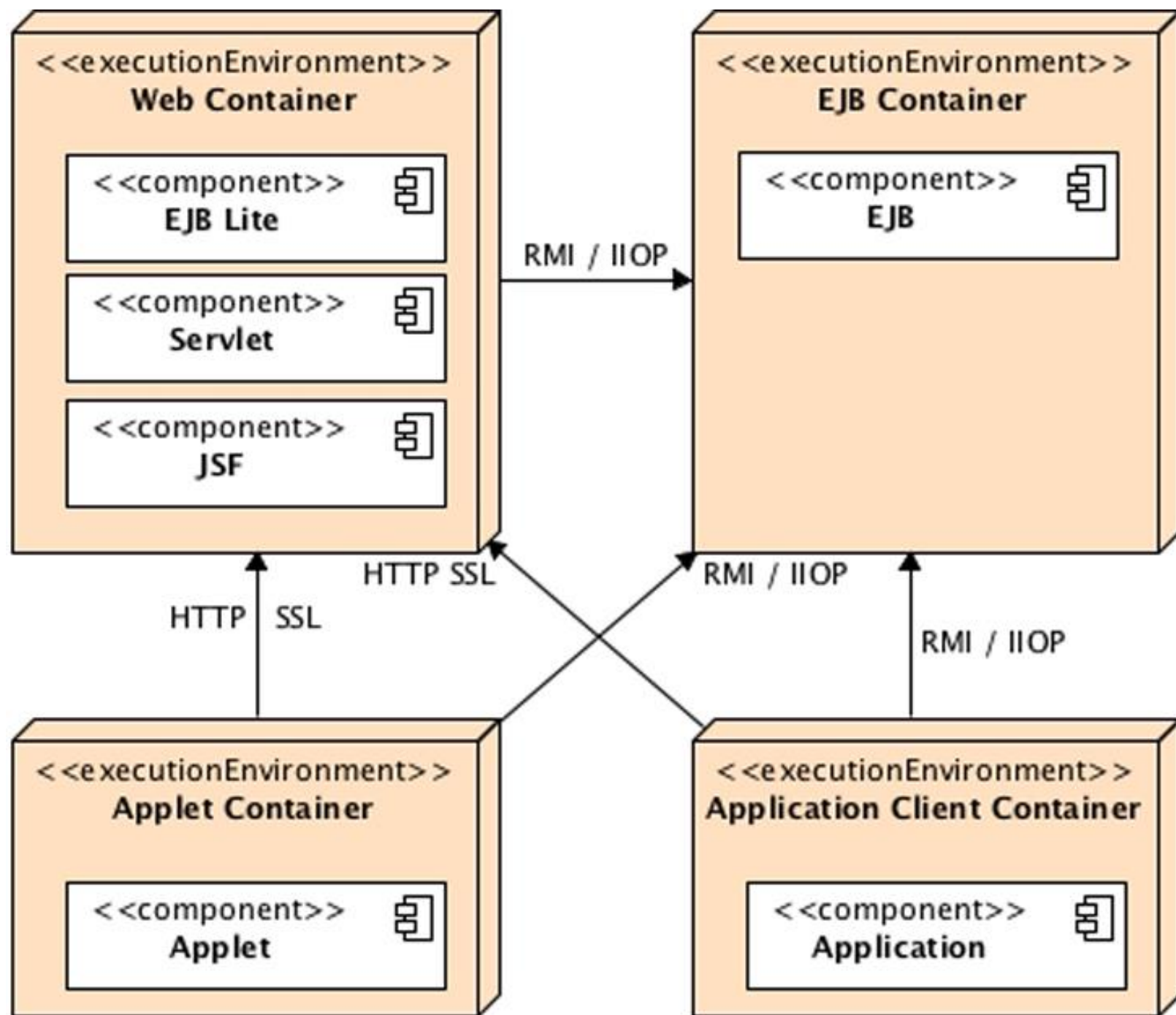
❑ Web Services

- Java EE brinda soporte para servicios web SOAP (JAX-WS) y servicios web RESTful (JAX-RS)

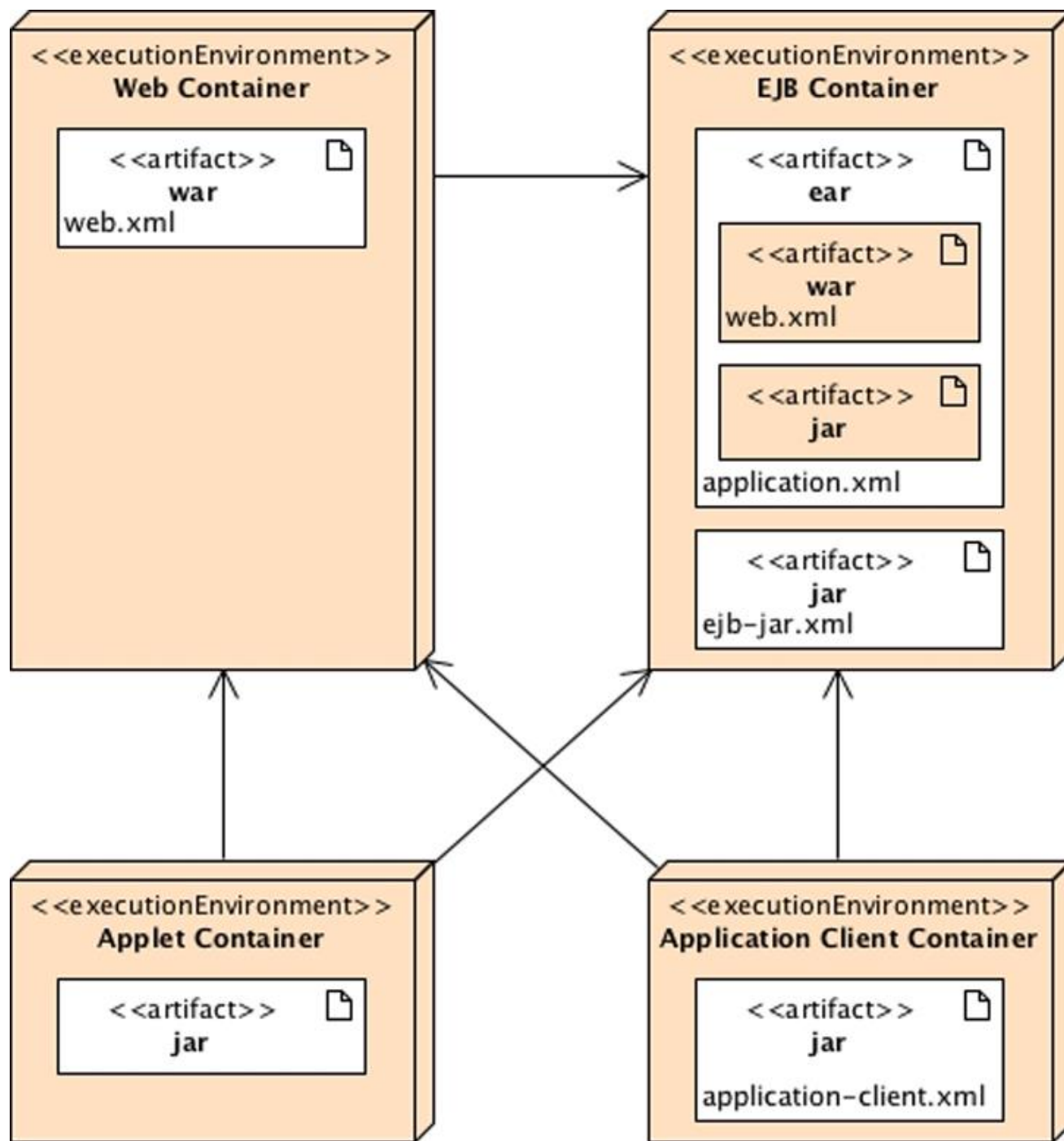
- ❑ Inyección de dependencias
 - Desde Java EE 5, algunos recursos (data sources, JMS factories, unidades de persistencia, EJBs) podían ser inyectados en componentes administrador
 - En Java EE 7 se va un paso mas al incorporar CDI (Context and Dependency Injection) como parte de la especificación



- ❑ Los componentes instalados en los contenedores, pueden ser invocados usando diferentes protocolos
 - HTTP: Es el protocolo usado por los componentes web para comunicarse con los clientes (browsers)
 - HTTPS: Es la combinación de HTTP y SSL
 - RMI-IIOP: RMI permite invocar remotamente métodos de objetos, sin importar el protocolo de comunicación usado



- ❑ Para poder ser instalados en un container, los componentes deben primero ser empaquetados siguiendo un formato predefinido
- ❑ Java define el formato JAR (Java ARchive)
 - Agregan múltiples archivos, como ser clases, descriptores, recursos, bibliotecas externas, etc
 - Se comprimen usando el formato ZIP
- ❑ Java EE extiende esta idea, definiendo diferentes tipos de archivos comprimidos

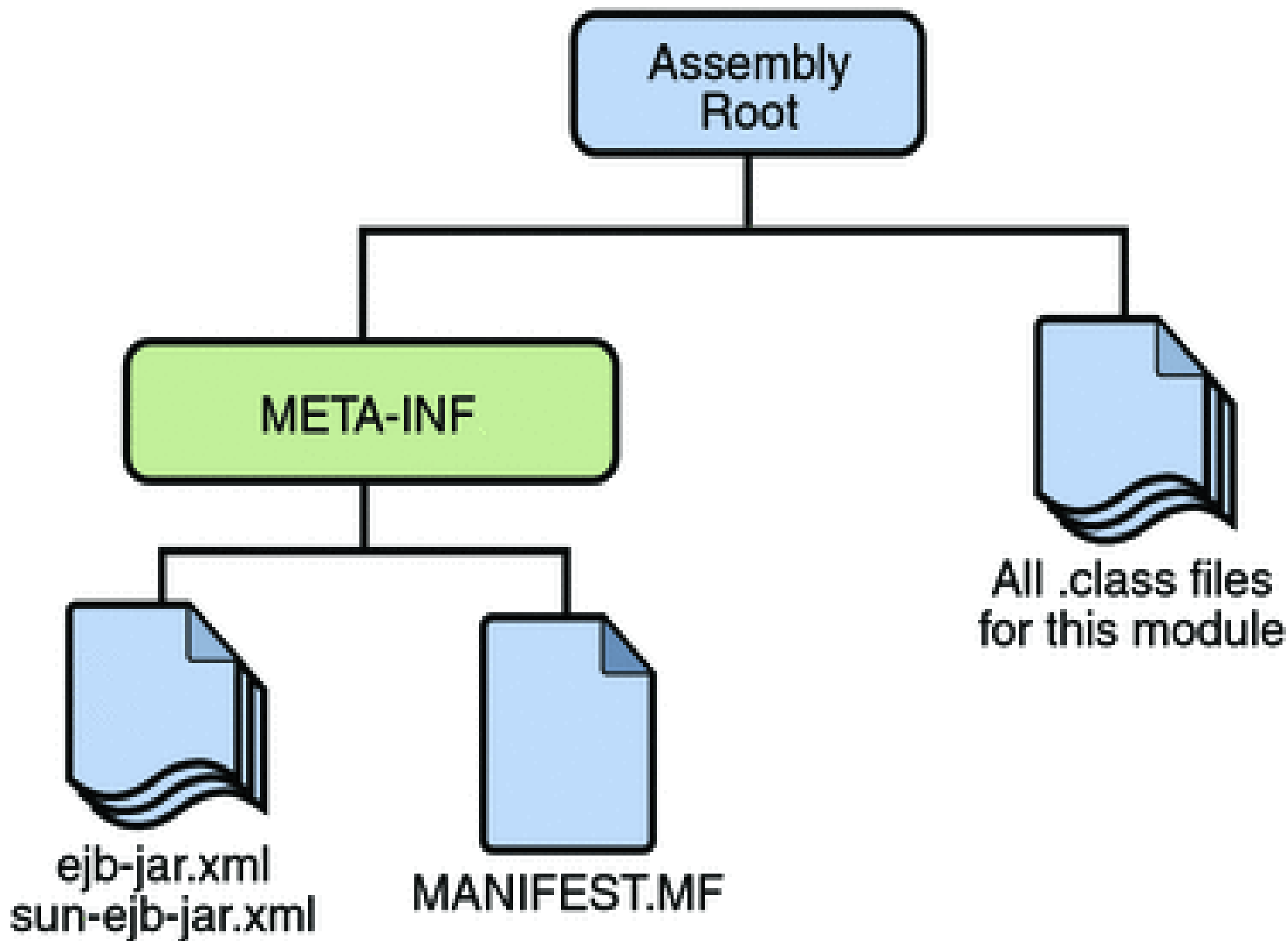


□ Application Client Module

- Contiene clases java y otros recursos empaquetados en un archivo JAR
- Puede ser ejecutado en un ambiente Java estándar (Java SE) o en un application client container
- Puede opcionalmente incluir un descriptor en META-INF/aplicacion-client.xml

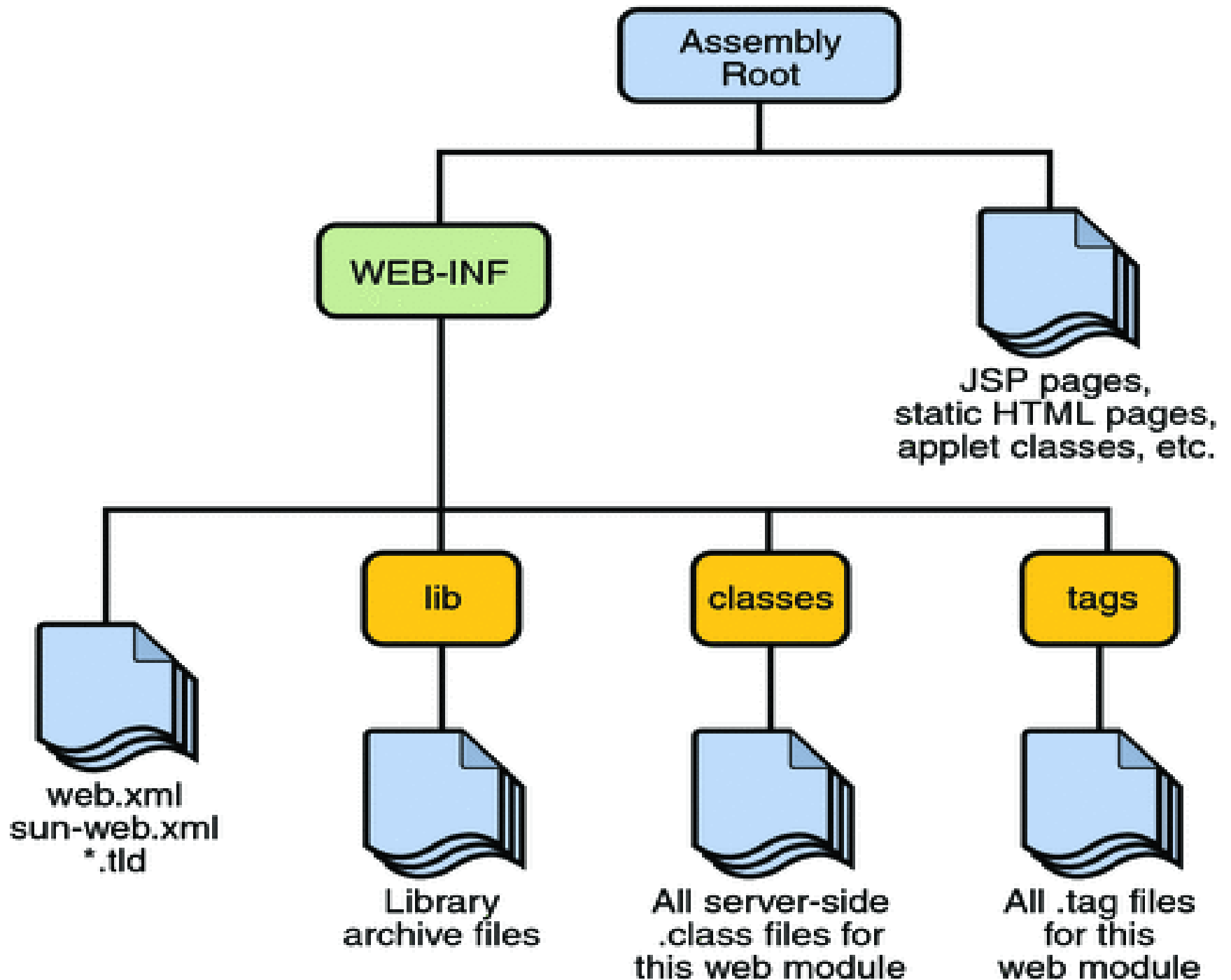
❑ EJB Module

- Contiene uno o mas componentes EJB (Session Beans o Message Driven Beans) empaquetados dentro de un JAR
- Opcionalmente puede incluir un descriptor en META-INF/ejb-jar.xml
- Solo puede ser desplegado en un contenedor EJB



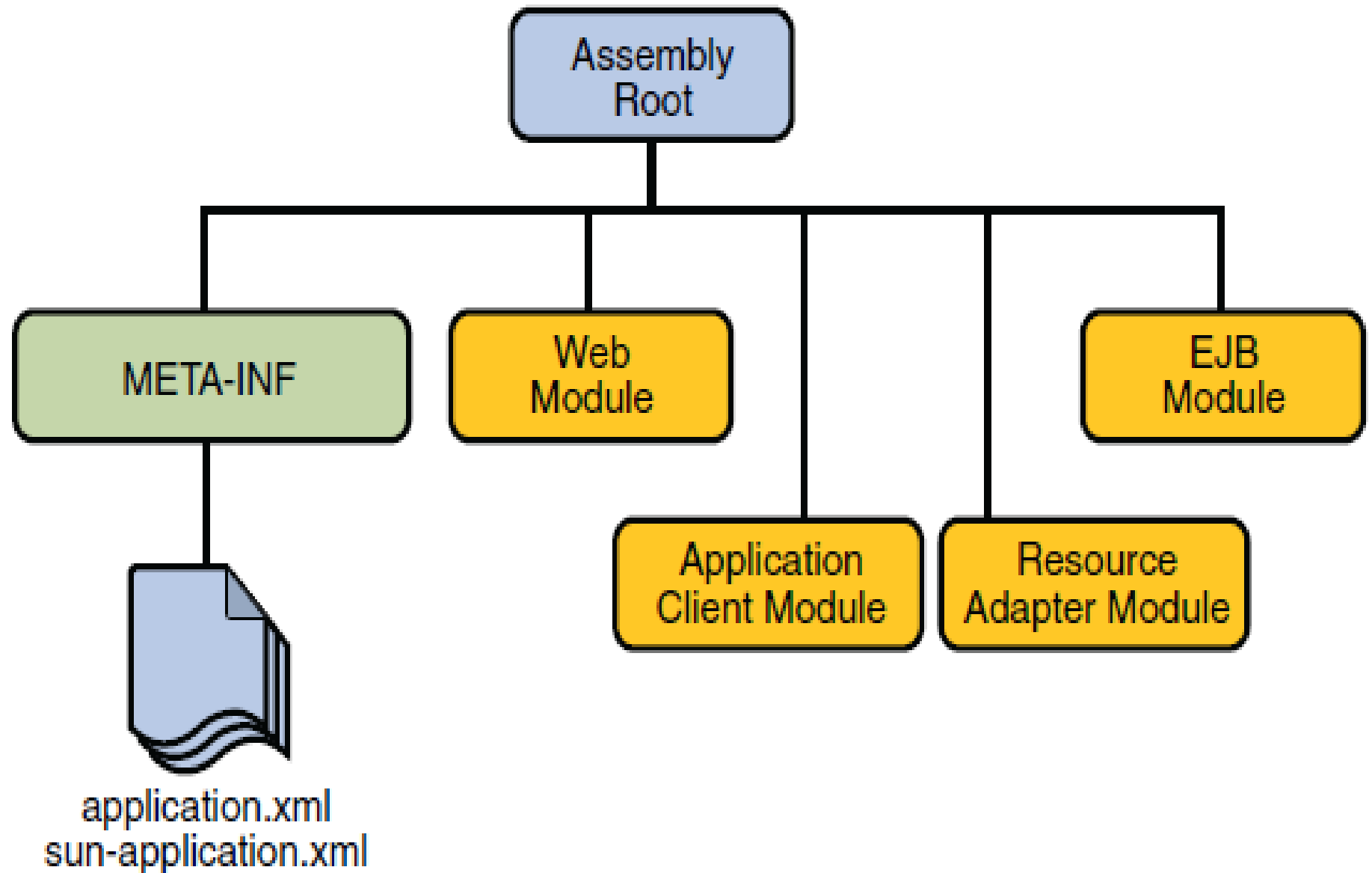
❑ Web Module

- Contiene componentes web (JSP, JSF, Servlets, etc.) así como componentes EJB lite
- Contiene también otros archivos relacionados con aplicaciones web (paginas HTML, imágenes, archivos Javascript)
- Los artefactos son empaquetados en un archivo con extensión .WAR
- Contiene opcionalmente un descriptor WEB-INF/web.xml



❑ Enterprise Archive

- Puede contener cero o mas módulos web, cero o mas módulos ejb, y otras bibliotecas de soporte o comunes a los demás módulos
- Todo esto se empaqueta y comprime en un archivo con extensión .EAR
- Opcionalmente puede contener un descriptor en META-INF/application.xml



Anotaciones vs Descriptores

- ❑ A nivel de programación, tenemos dos enfoques para resolver los problemas
- ❑ Programación imperativa:
 - En esta se especifica el algoritmo para llegar a una meta
 - “Como tengo que hacer la tarea”
- ❑ Programación declarativa
 - En esta se define que es lo que se quiere hacer (pero no como)
 - “Quiero hacer esta tarea”

Anotaciones vs Descriptores

- ❑ En Java EE, el enfoque declarativo se logra a través del uso de metadatos
- ❑ Estos metadatos se pueden expresar de dos formas diferentes
 - Anotaciones: Metadatos embebidos en el código
 - Descriptores: Archivos (generalmente XML) que configuran el funcionamiento de la aplicación

Anotaciones vs Descriptores

- ❑ Desde Java EE 5 (con la introducción del concepto de anotación en Java), las anotaciones han proliferado en el código de las aplicaciones
- ❑ Decoran las construcciones Java con metadatos sobre el programa
- ❑ Por ejemplo, un POJO (Plain Old Java Object) puede ser transformado en un EJB a través del uso de anotaciones...

Anotaciones vs Descriptores

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB
implements ItemLocal, ItemRemote {

    @PersistenceContext(unitName = "PU_Persistencia")
    private EntityManager em;
    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }
}
```

Anotaciones vs Descriptores

- ❑ La otra forma de declarar lo anterior, es a través de un archivo XML, el deployment descriptor
- ❑ Se despliega junto con el componente dentro del contenedor
- ❑ El contenedor accede a este archivo para obtener información de que es lo que se esta tratando de instalar en el contenedor

Anotaciones vs Descriptores

```
<ejb-jar xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/ejb-jar 3 2.xsd"
  version="3.2">
  <enterprise-beans>
    <session>
      <ejb-name>ItemEJB</ejb-name>
      <remote>org.test.ItemRemote</remote>
      <local>org.test.ItemLocal</local>
      <local-bean/>
      <ejb-class>org.test.ItemEJB</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

File	Specification	Paths
application.xml	Java EE	META-INF
application-client.xml	Java EE	META-INF
beans.xml	CDI	META-INF or WEB-INF
ra.xml	JCA	META-INF
ejb-jar.xml	EJB	META-INF or WEB-INF
faces-config.xml	JSF	WEB-INF
persistence.xml	JPA	META-INF
validation.xml	Bean Validation	META-INF or WEB-INF
web.xml	Servlet	WEB-INF
web-fragment.xml	Servlet	WEB-INF
webservices.xml	SOAP Web Services	META-INF or WEB-INF

Anotaciones vs Descriptores

- ❑ Desde Java EE 5 la mayoría de los DD son opcionales, pudiendo trabajar con anotaciones en su lugar
- ❑ Sin embargo, podemos combinar lo mejor de los dos mundos
 - Las anotaciones son mas sencillas de escribir y adjuntar al código Java
 - Pero los archivos XML pueden ser modificados sin necesidad de recompilar el código

Anotaciones vs Descriptores

- ❑ En caso de trabajar con ambos, si existe un elemento que es configurado a la vez por una anotación y por un deployment descriptor, entonces el descriptor tiene precedencia a la anotación
- ❑ Lo definido en los archivos XML, le hace override a los metadatos definidos con anotaciones

Anotaciones vs Descriptores

- ❑ Hoy día el uso de anotaciones es preferido ante los descriptores
- ❑ **Esto es principalmente porque existe una tendencia a reemplazar un modelo de programación basado en dos lenguajes (Java + XML), por un modelo basado en un solo lenguaje (Java)**

Anotaciones vs Descriptores

- ❑ Java EE trabaja con la noción de programación por excepción
 - También conocido como **Convention over Configuration**
- ❑ La mayoría del comportamiento común no requiere ser configurado con metadatos
 - El contenedor se hace cargo de todos los valores por defecto
 - Los metadatos brindan la excepción

Modelo de programación

- ❑ La mayoría de las especificaciones de Java EE 7 usan el mismo modelo de programación
 - Desarrollamos un POJO
 - Lo decoramos con metadatos (Anotaciones o XML)
 - Lo instalamos en un contenedor
- ❑ Gracias a los metadatos, el contenedor sabe que servicios aplicar al componente

JSF Backing Bean

```
@Named
public class BookController {
    @Inject
    private BookEJB bookEJB;
    private Book book = new Book();
    private List<Book> bookList = new ArrayList<Book>();

    public String doCreateBook() {
        book = bookEJB.createBook(book);
        bookList = bookEJB.findBooks();
        return "listBooks.xhtml";
    }

    // Getters, setters
}
```


Stateless EJB

```
@Stateless
public class BookEJB {
    @Inject
    private EntityManager em;

    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }
    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

RESTful Web Service

```
@Path("books")
public class BookResource {
    @Inject
    private EntityManager em;

    @GET
    @Produces({"application/xml", "application/json"})
    public List<Book> getAllBooks() {
        Query query = em.createNamedQuery("findAllBooks");
        List<Book> books = query.getResultList();
        return books;
    }
}
```

- ❑ Es importante tener presente que Java EE es un súper conjunto de Java SE
- ❑ Esto implica que las mejoras y cambios en el mismo, impactan indirectamente en Java EE
 - Autoboxing, anotaciones, generics, enumeraciones, diagnósticos, gestión, monitoreo, invocación de lenguajes de scripting, soporte para lenguajes dinámicos, etc.
- ❑ Algunas mejoras interesantes...

□ String case

- Antes el operador “case” solo podía ser usado con números (integrales)
- A partir de ahora se aceptan los strings
- Evita el uso de cascadas de IF/THEN/ELSE

```
String action = "update";
switch (action) {
    case "create":
        create();
        break;
    case "read":
        read();
        break;
    case "update":
        update();
        break;
    case "delete":
        delete();
        break;
    default:
        noCrudAction(action);
}
```

❑ Diamond

- Disminuye la verborragia en el uso de generics
- No es necesario para la implementación declarar el mismo tipado genérico que para la variable

```
// Sin diamond
List<String> list = new ArrayList<String>();
Map<Reference<Object>, Map<Integer, List<String>>> map =
    new HashMap<Reference<Object>, Map<Integer, List<String>>>();
```

```
// Con diamond
List<String> list = new ArrayList<>();
Map<Reference<Object>, Map<Integer, List<String>>> map = new HashMap<>();
```

□ Try con recursos

- Permite declarar que un bloque try/catch/finally utilizara una ser de recursos que deben ser liberados una vez finalizado el uso del bloque
- Disminuye mucho el volumen de código necesario para escribir un bloque

```
try {  
    InputStream input = new FileInputStream(in.txt);  
    try {  
        OutputStream output = new FileOutputStream(out.txt);  
        try {  
            byte[] buf = new byte[1024];  
            int len;  
            while ((len = input.read(buf)) >= 0)  
                output.write(buf, 0, len);  
        } finally {  
            output.close();  
        }  
    } finally {  
        input.close();  
    }  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

```
try (InputStream input = new FileInputStream(in.txt);  
    OutputStream output = new FileOutputStream(out.txt)) {  
  
    byte[] buf = new byte[1024];  
    int len;  
  
    while ((len = input.read(buf)) >= 0)  
        output.write(buf, 0, len);  
  
} catch (IOException e) {  
    e.printStackTrace();  
}
```


- ❑ Multicatch: Disminuye el volumen de código escrito

```
try {  
    // Do something  
  
} catch(SAXException e) {  
    e.printStackTrace();  
} catch(IOException e) {  
    e.printStackTrace();  
} catch(ParserConfigurationException e) {  
    e.printStackTrace();  
}
```

```
try {  
    // Do something  
} catch( SAXException |  
        IOException |  
        ParserConfigurationException e) {  
    e.printStackTrace();  
}
```

□ NIO.2

- Se provee un nuevo package para el manejo de archivos: **java.nio**
- Tiene una sintaxis mas expresiva
- Tiene el objetivo de reemplazar el uso de **java.io**
- Mejora el manejo de excepciones, tiene una interfaz mas clara y sencilla, provee nuevas facilidades a la hora de acceder al filesystem (links simbólicos, noción de partición, etc)

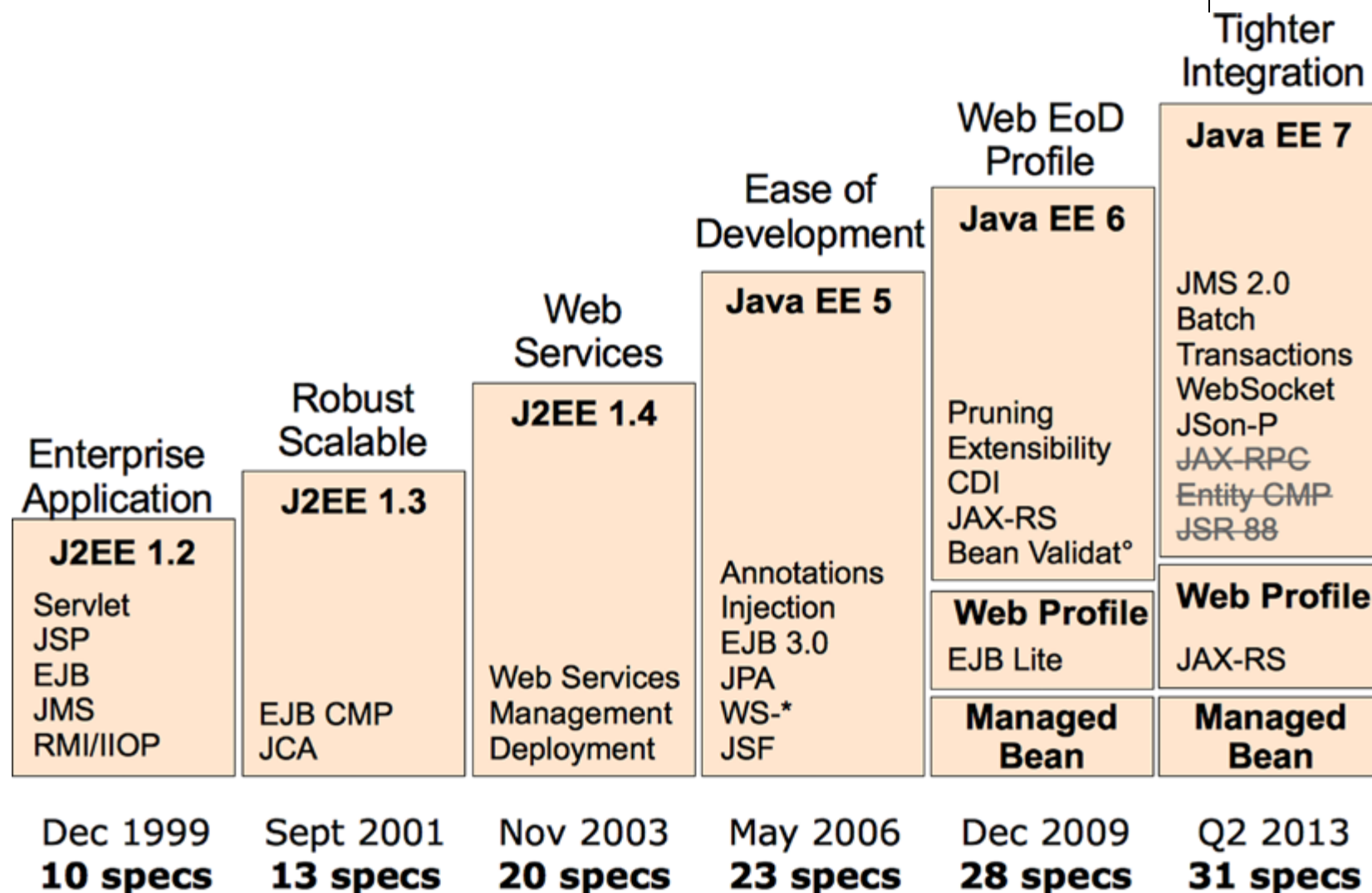
```
Path path = Paths.get("source.txt");
boolean exists = Files.exists(path);
boolean isDirectory = Files.isDirectory(path);
boolean isExecutable = Files.isExecutable(path);
boolean isHidden = Files.isHidden(path);
boolean isReadable = Files.isReadable(path);
boolean isRegularFile = Files.isRegularFile(path);
boolean isWritable = Files.isWritable(path);
long size = Files.size(path);

// Copies a file
Files.copy(Paths.get("source.txt"), Paths.get("dest.txt"));

// Reads a text file
List<String> lines = Files.readAllLines(Paths.get("source.txt"), UTF_8);
for (String line : lines) {
    System.out.println(line);
}

// Deletes a file
Files.delete(path);
```

14 años de Java EE



Especificaciones de Java EE 7

Specification	Version	JSR	URL
Java EE	7.0	342	http://jcp.org/en/jsr/detail?id=342
Web Profile	7.0	342	http://jcp.org/en/jsr/detail?id=342
Managed Beans	1.0	316	http://jcp.org/en/jsr/detail?id=316

Specification	Version	JSR	URL
JAX-WS	2.2a	224	http://jcp.org/en/jsr/detail?id=224
JAXB	2.2	222	http://jcp.org/en/jsr/detail?id=222
Web Services	1.3	109	http://jcp.org/en/jsr/detail?id=109
Web Services Metadata	2.1	181	http://jcp.org/en/jsr/detail?id=181
JAX-RS	2.0	339	http://jcp.org/en/jsr/detail?id=339
JSON-P	1.0	353	http://jcp.org/en/jsr/detail?id=353

Especificaciones de Java EE 7

Specification	Version	JSR	URL
JSF	2.2	344	http://jcp.org/en/jsr/detail?id=344
JSP	2.3	245	http://jcp.org/en/jsr/detail?id=245
Debugging Support for Other Languages	1.0	45	http://jcp.org/en/jsr/detail?id=45
JSTL	1.2	52	http://jcp.org/en/jsr/detail?id=52
Servlet	3.1	340	http://jcp.org/en/jsr/detail?id=340
WebSocket	1.0	356	http://jcp.org/en/jsr/detail?id=356
Expression Language	3.0	341	http://jcp.org/en/jsr/detail?id=341

Especificaciones de Java EE 7

Specification	Version	JSR	URL
EJB	3.2	345	http://jcp.org/en/jsr/detail?id=345
Interceptors	1.2	318	http://jcp.org/en/jsr/detail?id=318
JavaMail	1.5	919	http://jcp.org/en/jsr/detail?id=919
JCA	1.7	322	http://jcp.org/en/jsr/detail?id=322
JMS	2.0	343	http://jcp.org/en/jsr/detail?id=343
JPA	2.1	338	http://jcp.org/en/jsr/detail?id=338
JTA	1.2	907	http://jcp.org/en/jsr/detail?id=907

Especificaciones de Java EE 7

Specification	Version	JSR	URL
JACC	1.4	115	http://jcp.org/en/jsr/detail?id=115
Bean Validation	1.1	349	http://jcp.org/en/jsr/detail?id=349
Contexts and Dependency Injection	1.1	346	http://jcp.org/en/jsr/detail?id=346
Dependency Injection for Java	1.0	330	http://jcp.org/en/jsr/detail?id=330
Batch	1.0	352	http://jcp.org/en/jsr/detail?id=352
Concurrency Utilities for Java EE	1.0	236	http://jcp.org/en/jsr/detail?id=236
Java EE Management	1.1	77	http://jcp.org/en/jsr/detail?id=77
Java Authentication Service Provider Interface for Containers	1.0	196	http://jcp.org/en/jsr/detail?id=196

Especificaciones de Java EE 7

❑ Relacionadas fuertemente a Java SE 7

Specification	Version	JSR	URL
Common Annotations	1.2	250	http://jcp.org/en/jsr/detail?id=250
JDBC	4.1	221	http://jcp.org/en/jsr/detail?id=221
JNDI	1.2		
JAXP	1.3	206	http://jcp.org/en/jsr/detail?id=206
StAX	1.0	173	http://jcp.org/en/jsr/detail?id=173
JAAS	1.0		
JMX	1.2	3	http://jcp.org/en/jsr/detail?id=3
JAXB	2.2	222	http://jcp.org/en/jsr/detail?id=222
JAF	1.1	925	http://jcp.org/en/jsr/detail?id=925
SAAJ	1.3		http://java.net/projects/saaJ

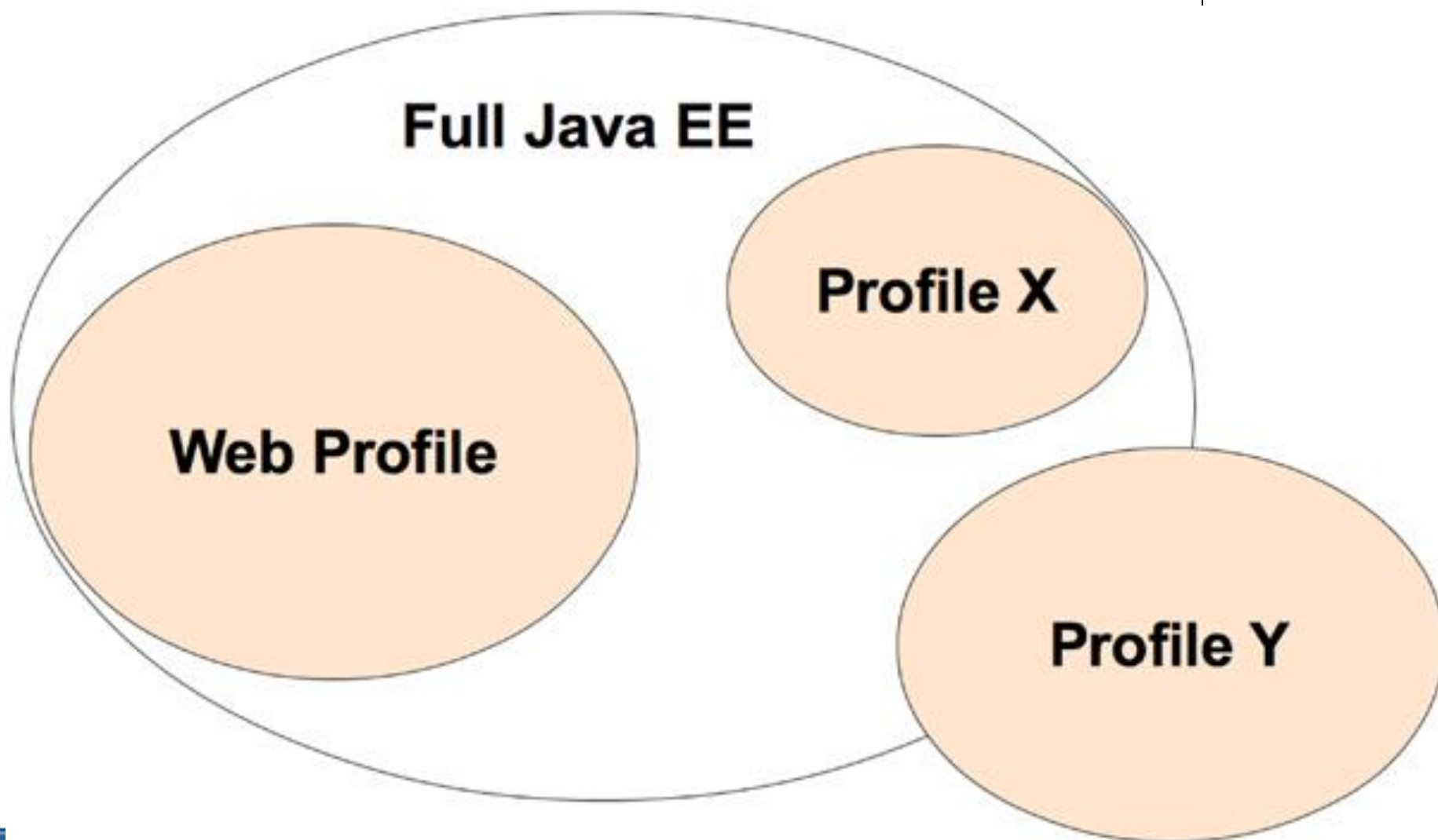
Web Profile 7

- ❑ Los perfiles se introdujeron en la versión 6 de Java EE
- ❑ El objetivo principal es disminuir el tamaño de la especificación para satisfacer mejor las necesidades del desarrollador
- ❑ **No importa el tamaño de la aplicación Java EE 7 que desarrollemos, siempre la vamos a instalar en un servidor que brinda servicios y APIs para 31 especificaciones**

Web Profile 7

- ❑ Una de las mayores críticas entonces a Java EE, es que es demasiado grande
- ❑ Para esto se crearon los perfiles
- ❑ El objetivo central es armar subconjuntos (o superconjuntos) de especificaciones que se adapten mejor a las necesidades del desarrollador

Web Profile 7



Especificaciones de Web Profile 7

Specification	Version	JSR	URL
JSF	2.2	344	http://jcp.org/en/jsr/detail?id=344
JSP	2.3	245	http://jcp.org/en/jsr/detail?id=245
JSTL	1.2	52	http://jcp.org/en/jsr/detail?id=52
Servlet	3.1	340	http://jcp.org/en/jsr/detail?id=340
WebSocket	1.0	356	http://jcp.org/en/jsr/detail?id=356
Expression Language	3.0	341	http://jcp.org/en/jsr/detail?id=341
EJB Lite	3.2	345	http://jcp.org/en/jsr/detail?id=345
JPA	2.1	338	http://jcp.org/en/jsr/detail?id=338
JTA	1.2	907	http://jcp.org/en/jsr/detail?id=907
Bean Validation	1.1	349	http://jcp.org/en/jsr/detail?id=349

Especificaciones de Web Profile 7

Specification	Version	JSR	URL
Managed Beans	1.0	316	http://jcp.org/en/jsr/detail?id=316
Interceptors	1.2	318	http://jcp.org/en/jsr/detail?id=318
Contexts and Dependency Injection	1.1	346	http://jcp.org/en/jsr/detail?id=346
Dependency Injection for Java	1.0	330	http://jcp.org/en/jsr/detail?id=330
Debugging Support for Other Languages	1.0	45	http://jcp.org/en/jsr/detail?id=45
JAX-RS	2.0	339	http://jcp.org/en/jsr/detail?id=339
JSON-P	1.0	353	http://jcp.org/en/jsr/detail?id=353

Java EE

Un ejemplo

- ❑ Vamos a implementar una pequeña aplicación para gestión de proyectos
- ❑ Vamos a manejar un par de entidades, “Project” y “Task”
- ❑ Estas entidades van a estar mapeadas a la base de datos, siendo utilizadas luego por la lógica de negocio de nuestra aplicación

```
import org.hibernate.validator.constraints.NotEmpty;
import javax.persistence.*;
import javax.validation.constraints.Size;
import java.util.*;

@Entity
public class Project {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "PROJECT_ID") private Integer id;
    @NotEmpty @Size(max = 64)
    private String name;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "project", fetch = FetchType.EAGER)
    private List<Task> tasks = new ArrayList<>();

    public Project() { /* Required for JPA */ }
    public Project(String name) { this.name = name; }

    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public List<Task> getTasks() { return tasks; }
    public void setTasks(List<Task> tasks) { this.tasks = tasks; }
```

```
import org.hibernate.validator.constraints.NotEmpty;
import javax.persistence.*;
import javax.validation.constraints.Size;
import java.util.*;

@Entity
public class Project {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "PROJECT_ID") private Integer id;
    @NotEmpty @Size(max = 64) private String name;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "project", fetch = FetchType.EAGER)
    private List<Task> tasks = new ArrayList<>();

    public Project() { /* Required for JPA */ }
    public Project(String name) { this.name = name; }

    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public List<Task> getTasks() { return tasks; }
    public void setTasks(List<Task> tasks) { this.tasks = tasks; }
```

```

public boolean addTask(Task task) {
    if (!tasks.contains(task)) {
        Project oldProject = task.getProject();
        if (oldProject != null) {
            removeTask(task);
        }
        tasks.add(task);
        return true;
    } else {
        return false;
    }
}

```

```

public boolean removeTask(Task task) {
    if (tasks.contains(task)) {
        tasks.remove(task);
        task.setProject(null);
        return true;
    } else {
        return false;
    }
}
// hashCode(), equals(), toString() omitted
}

```

```
import org.hibernate.validator.constraints.NotEmpty;
import javax.persistence.*;
import javax.validation.constraints.*;
import java.util.Date;
```

— @Entity

```
public class Task {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "TASK_ID") private Integer id;

    @NotEmpty @Size(max = 256) private String name;

    @Temporal(TemporalType.DATE)
    @Column(name = "TARGET_NAME") @Future
    private Date targetDate;

    private boolean completed;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "PROJECT_ID")
    private Project project;

    public Task() { /* Required by JPA */}
    public Task(String name, Date targetDate, boolean completed) {
        this.name = name;
        this.targetDate = targetDate;
        this.completed = completed;
    }
    // getters and setters
}
```

```
import org.hibernate.validator.constraints.NotEmpty;
import javax.persistence.*;
import javax.validation.constraints.*;
import java.util.Date;
```

```
@Entity
```

```
public class Task {
```

```
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    @Column(name = "TASK_ID") private Integer id;
```

```
    @NotEmpty @Size(max = 256) private String name;
```

```
    @Temporal(TemporalType.DATE)
```

```
    @Column(name = "TARGET_NAME") @Future
```

```
    private Date targetDate;
```

```
    private boolean completed;
```

```
    @ManyToOne(cascade = CascadeType.ALL)
```

```
    @JoinColumn(name = "PROJECT_ID")
```

```
    private Project project;
```

```
    public Task() { /* Required by JPA */}
```

```
    public Task(String name, Date targetDate, boolean completed) {
```

```
        this.name = name;
```

```
        this.targetDate = targetDate;
```

```
        this.completed = completed;
```

```
    }
```

```
    // getters and setters
```

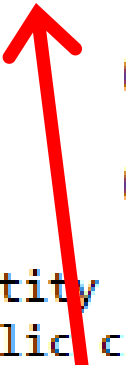
```
}
```

```
@NamedQueries({
    @NamedQuery(name = "Project.findAllProjects",
        query = "select p from Project p order by p.name"),
    @NamedQuery(name = "Project.findProjectById",
        query = "select p from Project p where p.id = :id"),
    @NamedQuery(name = "Project.findTaskById",
        query = "select t from Task t where t.id = :id"),})

@Entity
public class Project {/* ... */}
```

```
@NamedQueries({
    @NamedQuery(name = "Project.findAllProjects",
        query = "select p from Project p order by p.name"),
    @NamedQuery(name = "Project.findProjectById",
        query = "select p from Project p where p.id = :id"),
    @NamedQuery(name = "Project.findTaskById",
        query = "select t from Task t where t.id = :id"),})

@Entity
public class Project { /* ... */ }
```



Agregamos consultas escritas en JPQL

Las utilizamos desde los demás componentes de la aplicación

@Stateless

```
public class ProjectTaskService {
    @PersistenceContext
    private EntityManager entityManager;

    public void saveProject(Project project) {
        entityManager.persist(project);
    }
    public void updateProject(Project project ) {
        Project projectToBeUpdated = entityManager.merge(project);
        entityManager.persist(projectToBeUpdated);
    }
    public void removeProject(Project project) {
        Project projectToBeRemoved = entityManager.merge(project);
        entityManager.remove(projectToBeRemoved);
    }
    public List<Project> findAllProjects() {
        Query query =
            entityManager.createNamedQuery("Project.findAllProjects");
        return query.getResultList();
    }
    public List<Project> findProjectById(Integer id) {
        Query query =
            entityManager.createNamedQuery("Project.findProjectById")
                .setParameter("id", id );
        return query.getResultList();
    }
    public List<Task> findTaskById(Integer id) {
        Query query =
            entityManager.createNamedQuery("Project.findTaskById")
                .setParameter("id", id );
        return query.getResultList();
    }
}
```

```

@Stateless
public class ProjectTaskService {
    @PersistenceContext
    private EntityManager entityManager;

    public void saveProject(Project project) {
        entityManager.persist(project);
    }

    public void updateProject(Project project ) {
        Project projectToBeUpdated = entityManager.merge(project);
        entityManager.persist(projectToBeUpdated);
    }

    public void removeProject(Project project) {
        Project projectToBeRemoved = entityManager.merge(project);
        entityManager.remove(projectToBeRemoved);
    }

    public List<Project> findAllProjects() {
        Query query =
            entityManager.createNamedQuery("Project.findAllProjects");
        return query.getResultList();
    }

    public List<Project> findProjectById(Integer id) {
        Query query =
            entityManager.createNamedQuery("Project.findProjectById")
                .setParameter("id", id );
        return query.getResultList();
    }

    public List<Task> findTaskById(Integer id) {
        Query query =
            entityManager.createNamedQuery("Project.findTaskById")
                .setParameter("id", id );
        return query.getResultList();
    }
}

```

```

@ServerEndpoint("/sockets")
@Stateless
public class ProjectWebSocketServerEndpoint {
    static SimpleDateFormat FMT = new SimpleDateFormat("dd-MMM-yyyy");
    @Inject ProjectTaskService service;
    @OnMessage
    public String retrieveProjectAndTasks(String message) {
        int projectId = Integer.parseInt(message.trim());
        List<Project> projects = service.findProjectById(projectId);
        StringWriter swriter = new StringWriter();
        JsonGeneratorFactory factory = Json.createGeneratorFactory(
            new HashMap<String, Object>() {
                { put(JsonGenerator.PRETTY_PRINTING, true);}
            }
        );
        JsonGenerator generator = factory.createGenerator(swriter);
        generator.writeStartArray();
        for (Project project: projects) {
            generator.writeStartObject()
                .write("id", project.getId())
                .write("name", project.getName())
                .writeStartArray("tasks");

            for (Task task: project.getTasks()) {
                generator.writeStartObject()
                    .write("id", task.getId())
                    .write("name", task.getName())
                    .write("targetDate", task.getTargetDate() == null ? "" :
                        FMT.format(task.getTargetDate()))
                    .write("completed", task.isCompleted())
                    .writeEnd();
            }
            generator.writeEnd().writeEnd();
        }
        generator.writeEnd().close();
        return swriter.toString();
    }
}

```

```
@ServerEndpoint("/sockets")
@Stateless
public class ProjectWebSocketServerEndpoint {
    static SimpleDateFormat FMT = new SimpleDateFormat("dd-MMM-yyyy");
    @Inject ProjectTaskService service;
    @OnMessage
    public String retrieveProjectAndTasks(String message) {
        int projectId = Integer.parseInt(message.trim());
        List<Project> projects = service.findProjectById(projectId);
        StringWriter swriter = new StringWriter();
        JsonGeneratorFactory factory = Json.createGeneratorFactory(
            new HashMap<String, Object>() {
                { put(JsonGenerator.PRETTY_PRINTING, true); }
            }
        );
        JsonGenerator generator = factory.createGenerator(swriter);
        generator.writeStartArray();
        for (Project project: projects) {
            generator.writeStartObject()
                .write("id", project.getId())
                .write("name", project.getName())
                .writeStartArray("tasks");

            for (Task task: project.getTasks()) {
                generator.writeStartObject()
                    .write("id", task.getId())
                    .write("name", task.getName())
                    .write("targetDate", task.getTargetDate() == null ? "" :
                        FMT.format(task.getTargetDate()))
                    .write("completed", task.isCompleted())
                    .writeEnd();
            }
            generator.writeEnd().writeEnd();
        }
        generator.writeEnd().close();
        return swriter.toString();
    }
}
```

