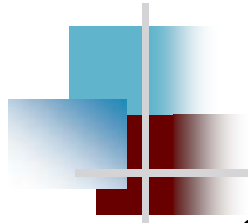


4.4. CBSE: desarrollo para y con reutilización

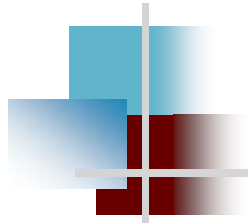


(Sommerville)



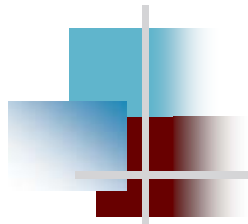
Procesos básicos CBSE

- CBSE (Component Based Software Engineering) tiene como objetivos desarrollar componentes y desarrollar sistemas a partir de esos componentes
- Desarrollo para reutilización
 - El **desarrollo de componentes** o servicios que se pueden reutilizar en otras aplicaciones.
 - Puede implicar la generalización de los componentes ya existentes.
- Desarrollo con reutilización
 - **Desarrollo** de nuevas aplicaciones **basado en componentes** y servicios existentes



a. Desarrollo para reutilización

- Construir componentes mediante generalización de los componentes (o partes de aplicaciones) existentes
- Un componente para reutilización:
 - Debe reflejar abstracciones estables de dominio
 - Debería ocultar la representación del estado
 - Debe ser lo más independiente posible
 - En caso de publicar excepciones, debe hacerlo a través de la interfaz de componente.
- Hay un equilibrio entre reutilización y facilidad de uso
 - Cuanto más general la interfaz, mayor capacidad de reutilización, pero será más compleja y por lo tanto menos fácil de utilizar.



Cambios

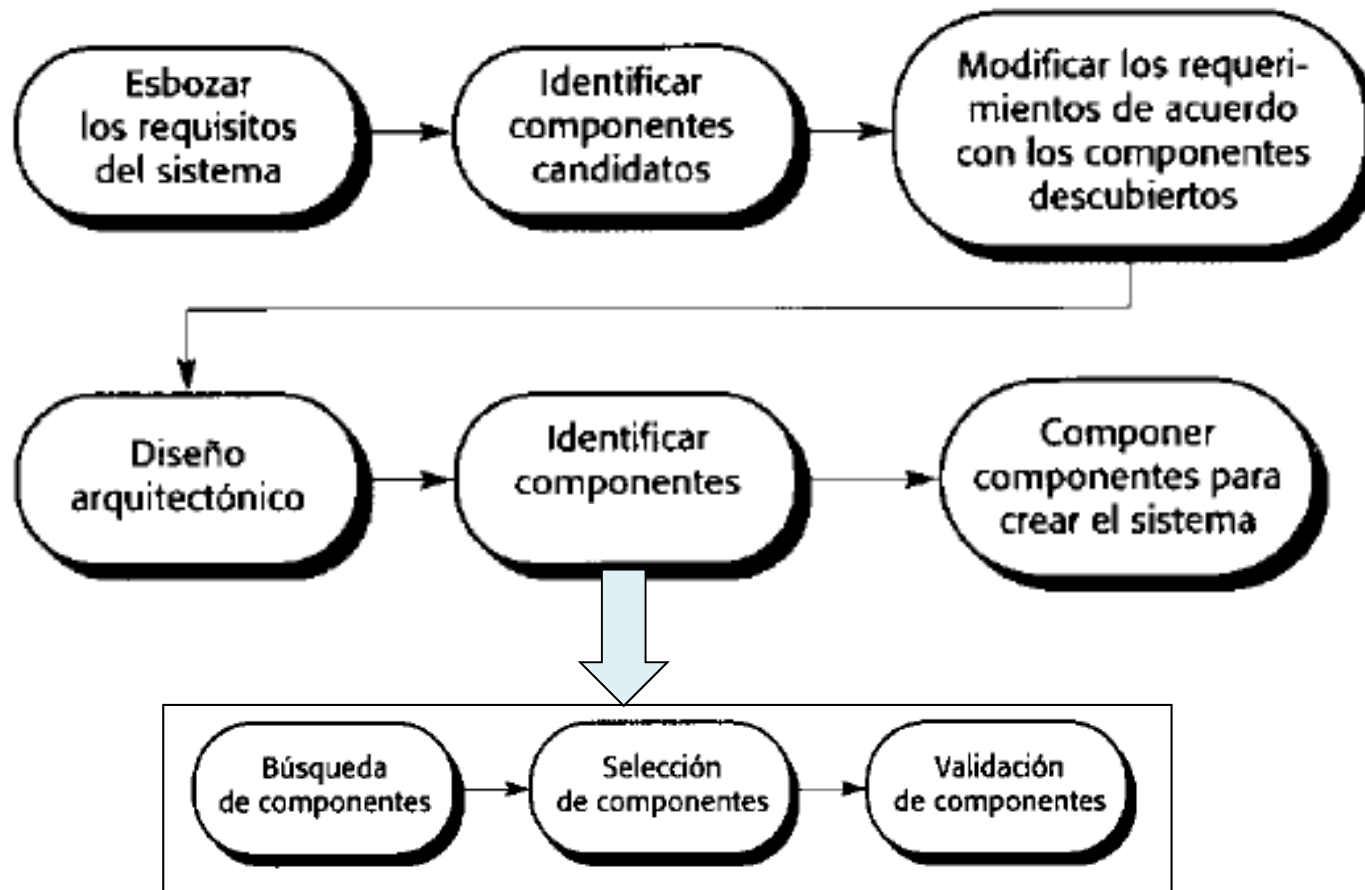
- Retirar los métodos específicos de la aplicación.
- Cambiar los nombres para hacerlos generales.
- Agregar métodos para ampliar la cobertura.
- Hacer un manejo de excepciones consistente.
- Integrar los componentes requeridos para reducir las dependencias.
- Agregar una interfaz de configuración para la adaptación de los componentes
 - (Ejemplo: Net Beans y paleta de JavaBeans)

Sistemas legados como componentes

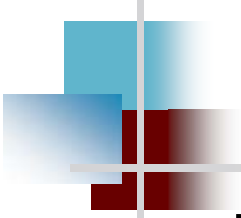


- Los sistemas existentes se pueden empaquetar como componentes para su reutilización.
- Requiere escribir un componente tipo envoltorio (wrapper) que implemente las interfaces para acceder al sistema anterior (y para que éste acceda a otros servicios)
 - Puede ser una solución mucho menos costosa que la reescritura del sistema completo.

b. Desarrollo con reutilización



Desarrollo Basado en Componentes

- 
- Encontrar e integrar componentes reutilizables.
 - Es indispensable buscar un equilibrio entre los requisitos ideales y los servicios prestados por los componentes disponibles.
 - (0) Determinar los requisitos -- no cambia
 - (1) La búsqueda (negociación incluida) de componentes que satisfagan los requisitos impuestos tanto por el cliente como por la arquitectura de la aplicación
 - (2) La evaluación de los componentes candidatos para seleccionar los más idóneos, incluyendo su Validación
 - (3) La adaptación y/o extensión de los componentes seleccionados para que se ajusten a los requisitos. Integración, configuración e interconexión de dichos componentes para construir la aplicación final



Identificación de componentes

■ Confianza

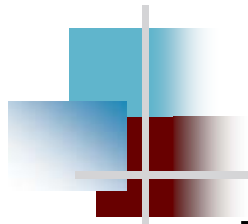
- Se debe poder confiar en el proveedor de un componente.
- Un componente puede que no funcione como se anuncia, y en el peor de los casos, puede violar la seguridad del sistema

■ Requisitos

- Diferentes componentes pueden satisfacer los diferentes requisitos de forma parcial

■ Validación

- La especificación del componente puede no ser lo suficientemente detallada para permitir pruebas necesarias
- Los componentes pueden tener una funcionalidad no deseada. ¿Cómo se puede probar que no va a interferir con mi aplicación? (fallo del Ariane 5!)



Validación de componentes

- Desarrollo de un conjunto de casos de prueba para un componente (o extender los casos de prueba suministrados)
- Desarrollo de una aplicación de prueba para ejecutar las pruebas
- El principal problema es que la especificación del componente no sea lo suficientemente detallada
- Además de probar que un componente hace lo que se necesita, hay que comprobar que el componente no incluye ningún código malicioso o bloquear la funcionalidad que no es necesaria



El caso del Ariane 5

- En 1996, la primera prueba de vuelo del cohete Ariane 5 terminó en desastre cuando el lanzador se salió de control 37 segundos después de despegar
- El problema se debió a un componente proveniente de una versión anterior del cohete lanzador (el sistema de navegación inercial) que fracasó porque la precondition necesaria para una función de ese componente no la cumplió el Ariane 5
- La funcionalidad que falló en este componente no se necesitaba en el Ariane 5!

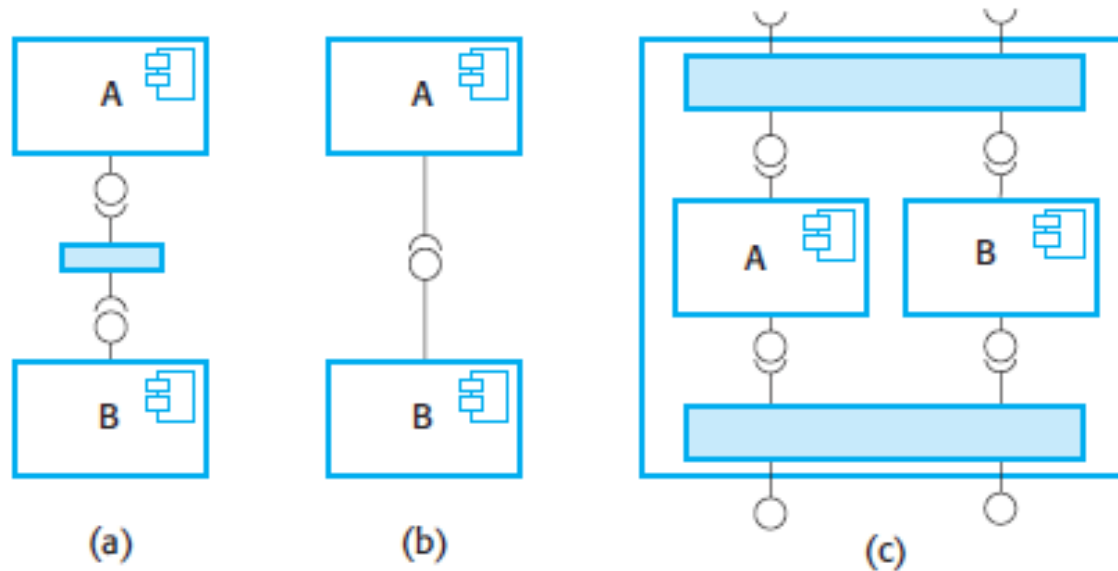
4.4.1.

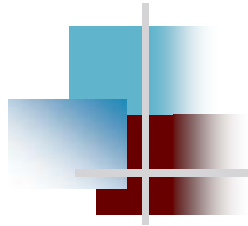
Composición y adaptación



Composición de componentes

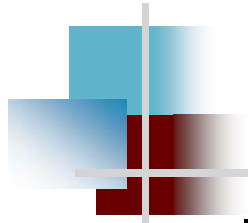
- Es el proceso de ensamblaje de componentes para crear un sistema.
- La composición involucra la integración de los componentes entre sí y con la infraestructura de componentes.
- Normalmente hay que escribir “código pegamento” (“glue code”) para integrar los componentes





Tipos de composición

- a: Composición secuencial
 - los componentes se ejecutan en secuencia. Esto implica componer las interfaces que proporciona cada componente.
- b: Composición jerárquica
 - uno de los componentes usa los servicios de otro. La interfaz proporcionada de un componente se conecta con la interfaz requerida de otro
- c: Composición aditiva
 - Las interfaces de dos componentes se unen para crear un nuevo componente. Las interfaces del componente integrado son una combinación de las interfaces de los componentes constituyentes.

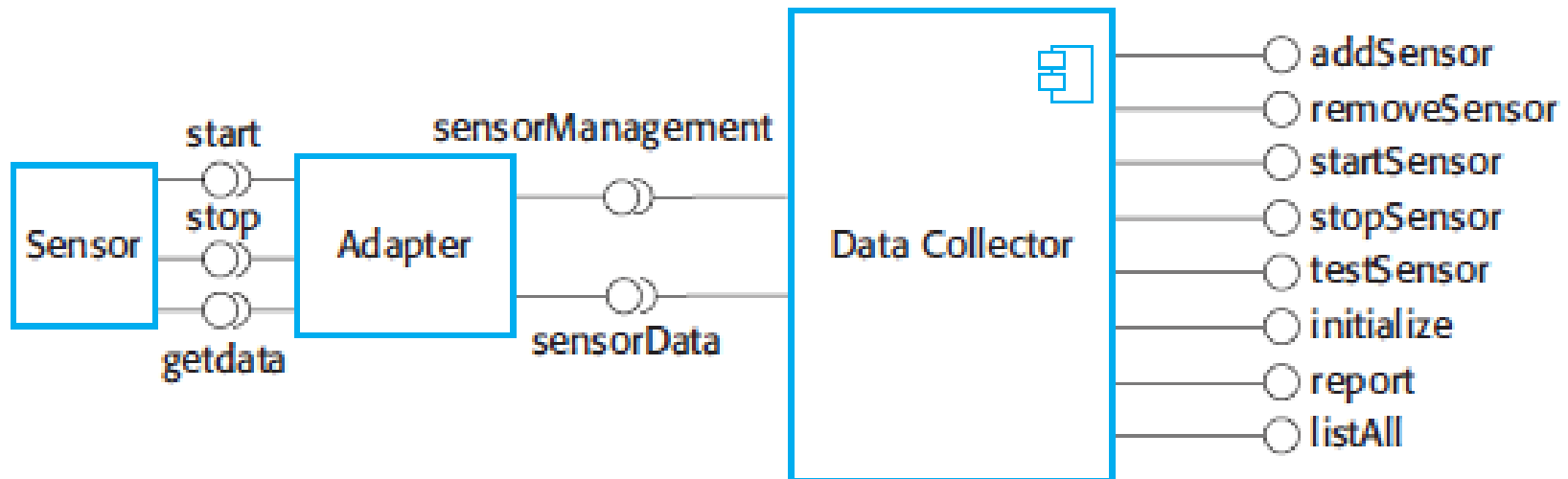


Compatibilidad de interfaces

- Incompatibilidad de parámetros
 - Las operaciones tienen el mismo nombre pero los parámetros o el resultado que devuelve son de tipos diferentes
- Incompatibilidad operaciones
 - Los nombres de las operaciones en las interfaces compuestas son diferentes
- Ausencia de operaciones
 - La interfaz que proporciona el interfaz de un componente es un subconjunto de la interfaz que requiere de otro

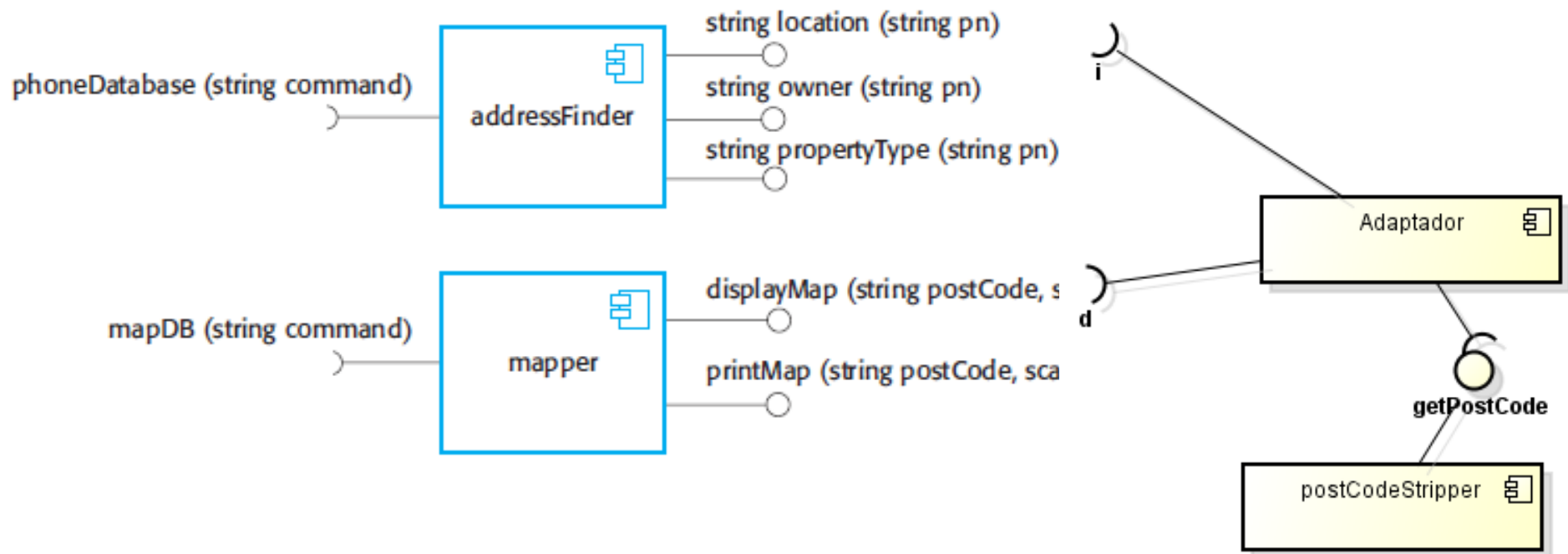
Ejemplos de adaptación

- Recolector de datos y sensor



Glue code

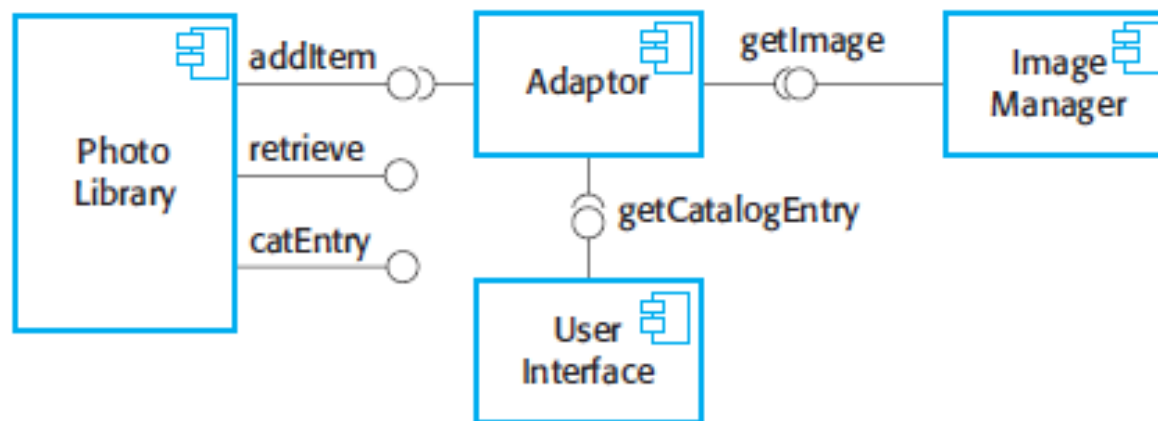
- Ej: adaptador que extrae el código postal de **addressFinder** con **postCodeStripper** y lo pasa al componente **mapper**



```
address = addressFinder.location (phonenumber) ;
postCode = postCodeStripper.getPostCode (address) ;
mapper.displayMap(postCode, 10000)
```


Adaptación y semántica

- Biblioteca de fotos





Semántica y sintaxis

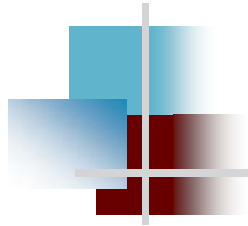
- Estudiar la documentación de los componentes para saber si interfaces que son sintácticamente compatibles son realmente compatibles:

```
/* Este método agrega una fotografía a la biblioteca y  
asocia el identificador de fotografía y descriptor de  
catálogo con la fotografía*/
```

```
public void addItem (Identifier pid, Photograph p,  
    CatalogEntry photodesc) ;
```

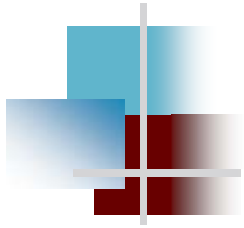
```
/* Otros... */
```

```
public Photograph retrieve (Identifier pid) ;  
public CatalogEntry catEntry (Identifier pid) ;
```



Semántica y sintaxis

- ¿qué pasa si el identificador de la fotografía ya está asociado con una fotografía en la biblioteca?
- Está la fotografía asociada con la entrada de catálogo,
 - si elimino la fotografía, ¿también se debe borrar la información del catálogo?"



OCL

```
public void addItem (Identifier pid , Photograph p,  
CatalogEntry photodesc) ;
```

```
context addItem(Identifier pid , Photograph p, CatalogEntry  
photodesc)
```

```
pre: PhotoLibrary.libSize() >= 0
```

```
PhotoLibrary.retrieve(pid) = null
```

```
post: libSize () = libSize()@pre + 1
```

```
PhotoLibrary.retrieve(pid) = p
```

```
PhotoLibrary.catEntry(pid) = photodesc
```

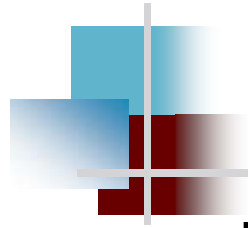
```
context deleteIdentifier (pid)
```

```
pre: PhotoLibrary.retrieve(pid) <> null
```

```
post: PhotoLibrary.retrieve(pid) = null
```

```
PhotoLibrary.libSize() = libSize()@pre-1
```

```
PhotoLibrary.catEntry(pid) = PhotoLibrary.catEntry(pid)@pre
```



Semántica (Expresiones OCL)

- La biblioteca debe existir
- No puede haber una fotografía en la biblioteca con el mismo identificador que la fotografía que se quiere incluir
- Cada entrada aumenta el tamaño de la biblioteca en un elemento
- Si se recupera utilizando el mismo identificador a continuación, se obtiene la foto que ha añadido
- Si se busca en el catálogo utilizando ese identificador, entonces devolverá la entrada de catálogo que acaba de insertar

4.5.

Diseño basado en Componentes (UML)

- Chesmann

Actividades de Diseño (enfoque Top Down)

- Diseño paso a paso...

Identificar Interfaces



Asignar responsabilidades a componentes



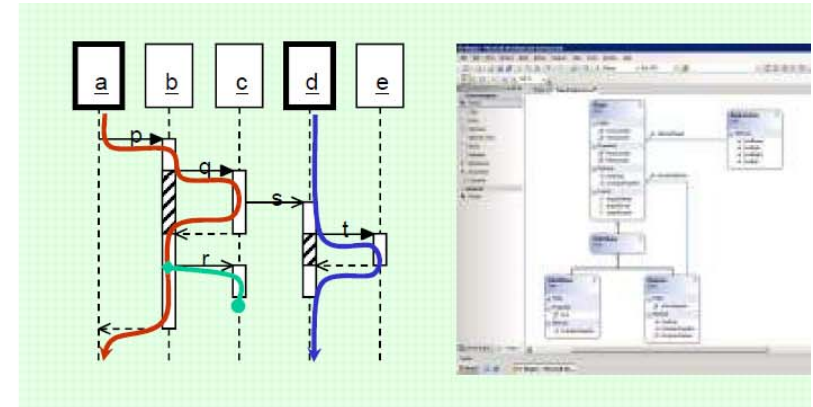
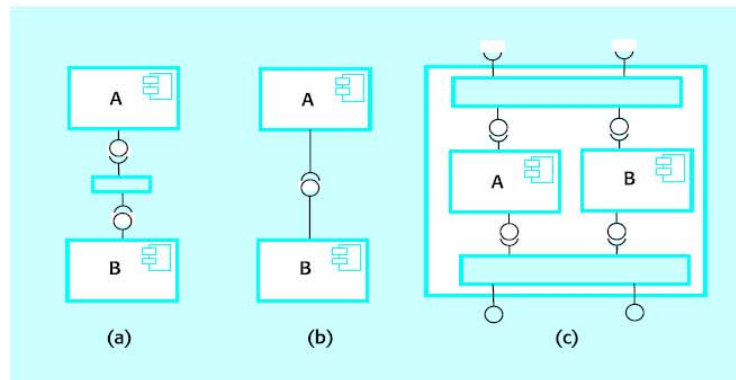
Diseñar las Interfaces

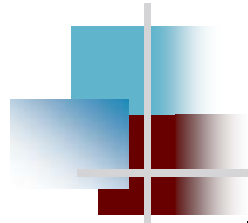


Verificar los requisitos



Comparar contra escenarios...



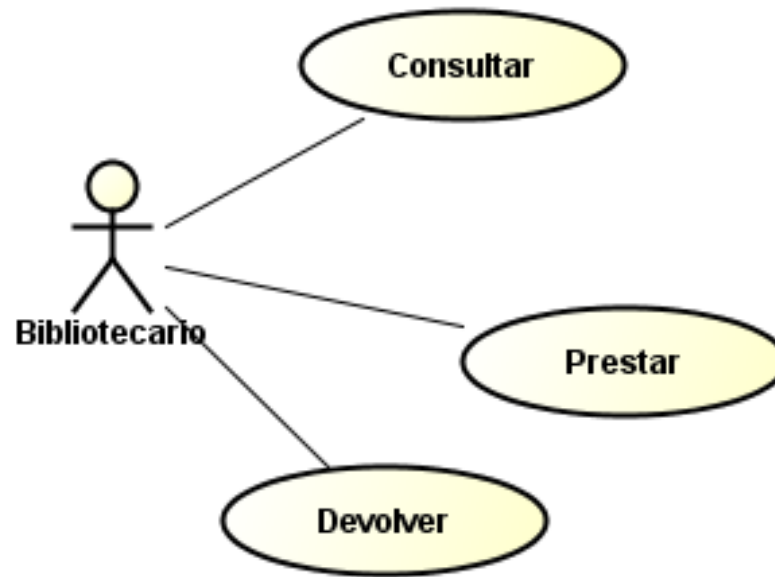


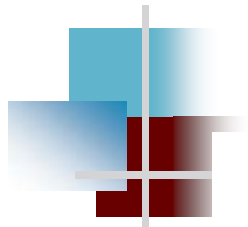
Actividades de diseño

- Identificar Interfaces ← Operaciones del Sistema
- Asignar responsabilidades (Interfaces) → componentes
- Diseñar en detalle las Interfaces
- Verificar los requisitos funcionales
- Comparar contra escenarios
- Analizar interacciones
- Analizar la Flexibilidad del sistema

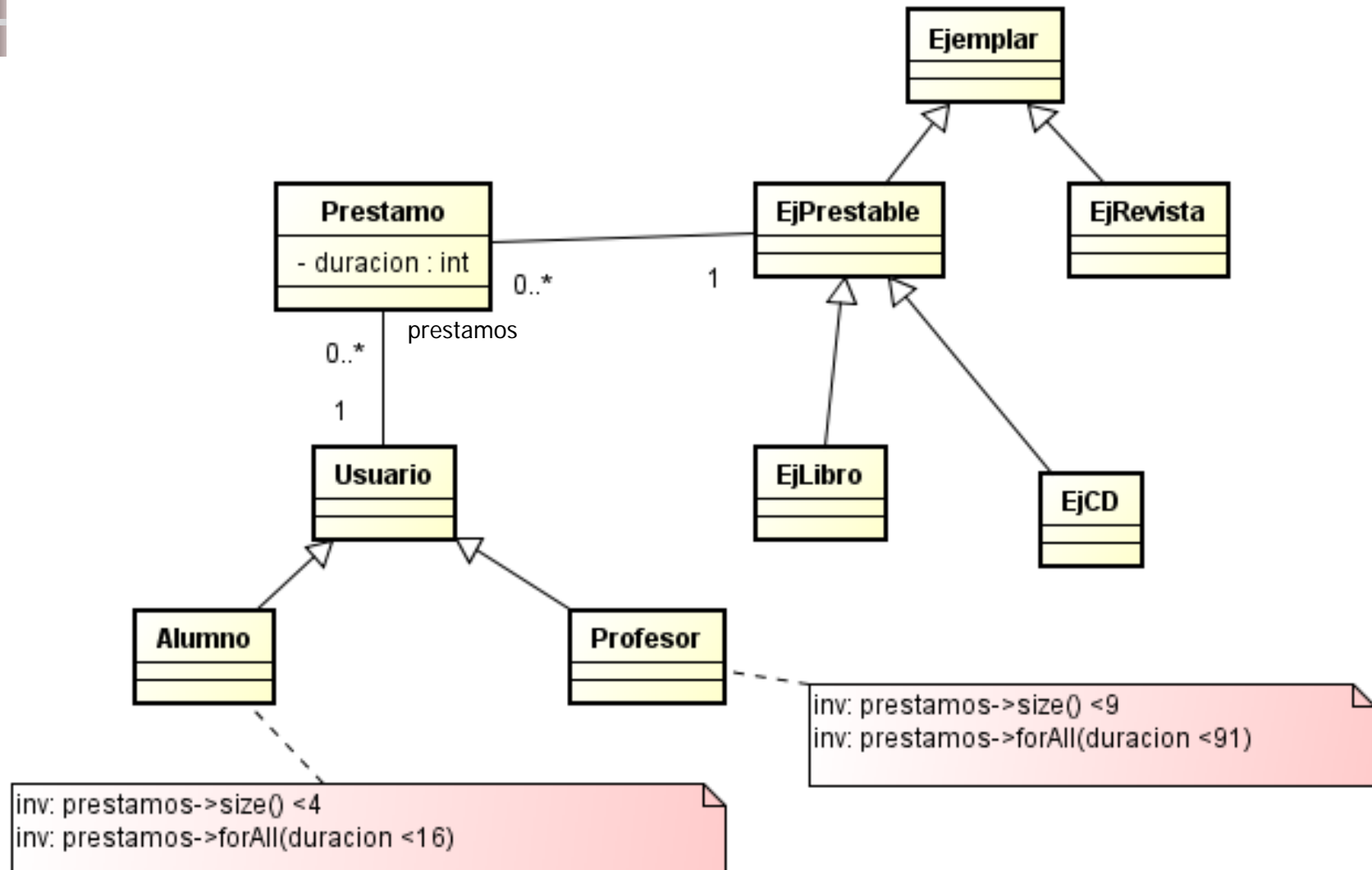
Ejemplo: Biblioteca

- Se pueden consultar (todos los ejemplares) o prestar ejemplares (de ciertos tipos)
- El préstamo es diferente para profesor o alumno

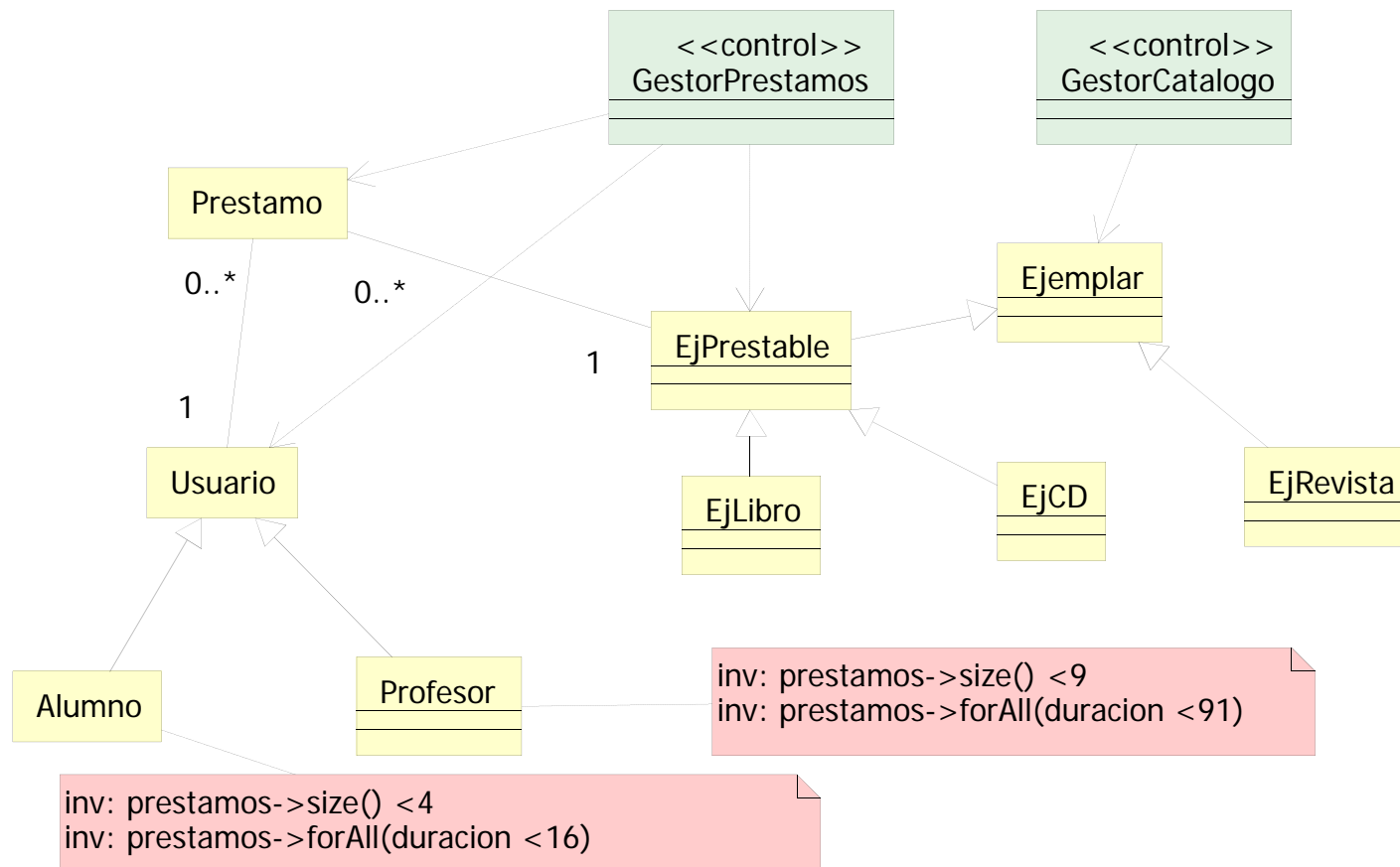




Dominio



Después DS y añadir operaciones: Clases <<Control>>

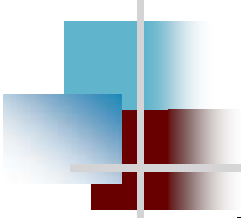




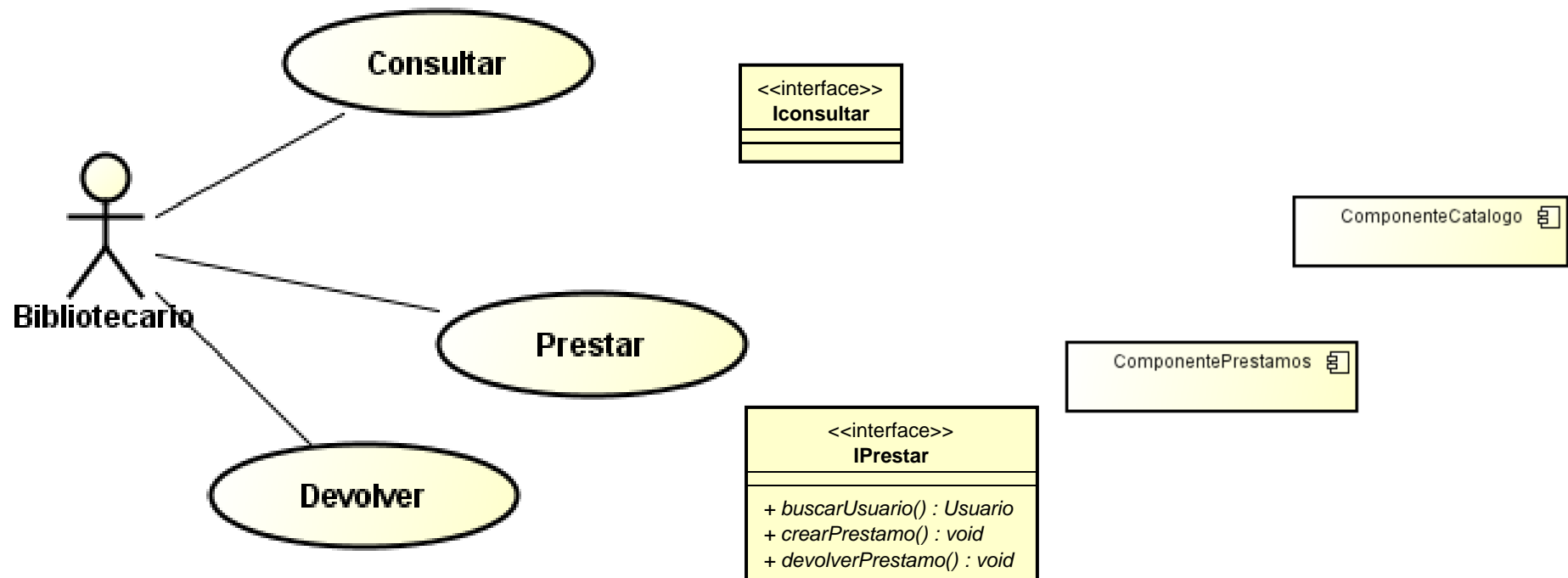
1. Identificar Interfaces

- Utilizar los requisitos funcionales para identificar responsabilidades (a partir de los casos de uso) de alto nivel
- La interfaces agrupan conjuntos de operaciones relacionadas
 - Información manejada
 - Servicios Ofrecidos

2. Asignar Responsabilidades a los componentes

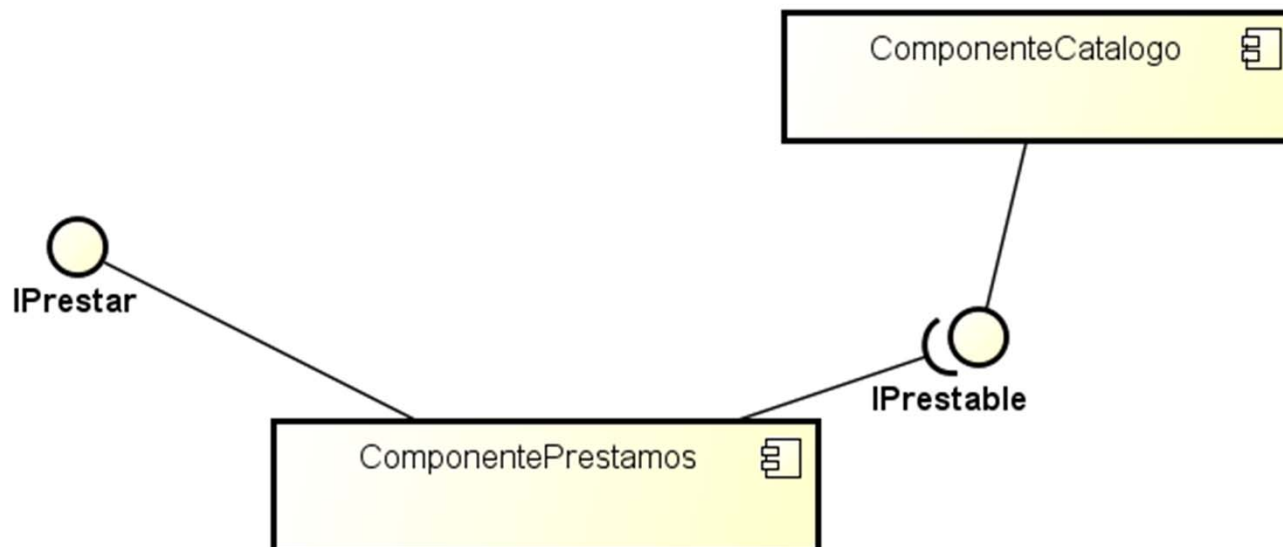
- 
- Después de identificar componentes candidatos, se les asignan responsabilidades claras (Interfaces)
 - Identificar los componentes que realizarán esas responsabilidades
 - Evaluar los componentes identificados contra los criterios de diseño deseables
 - Iterar sobre los pasos anteriores hasta obtener un conjunto de componentes
 - Prever interfaces requeridas para plug-in alternativos (características o algoritmos)

Componentes básicos



Responsabilidades

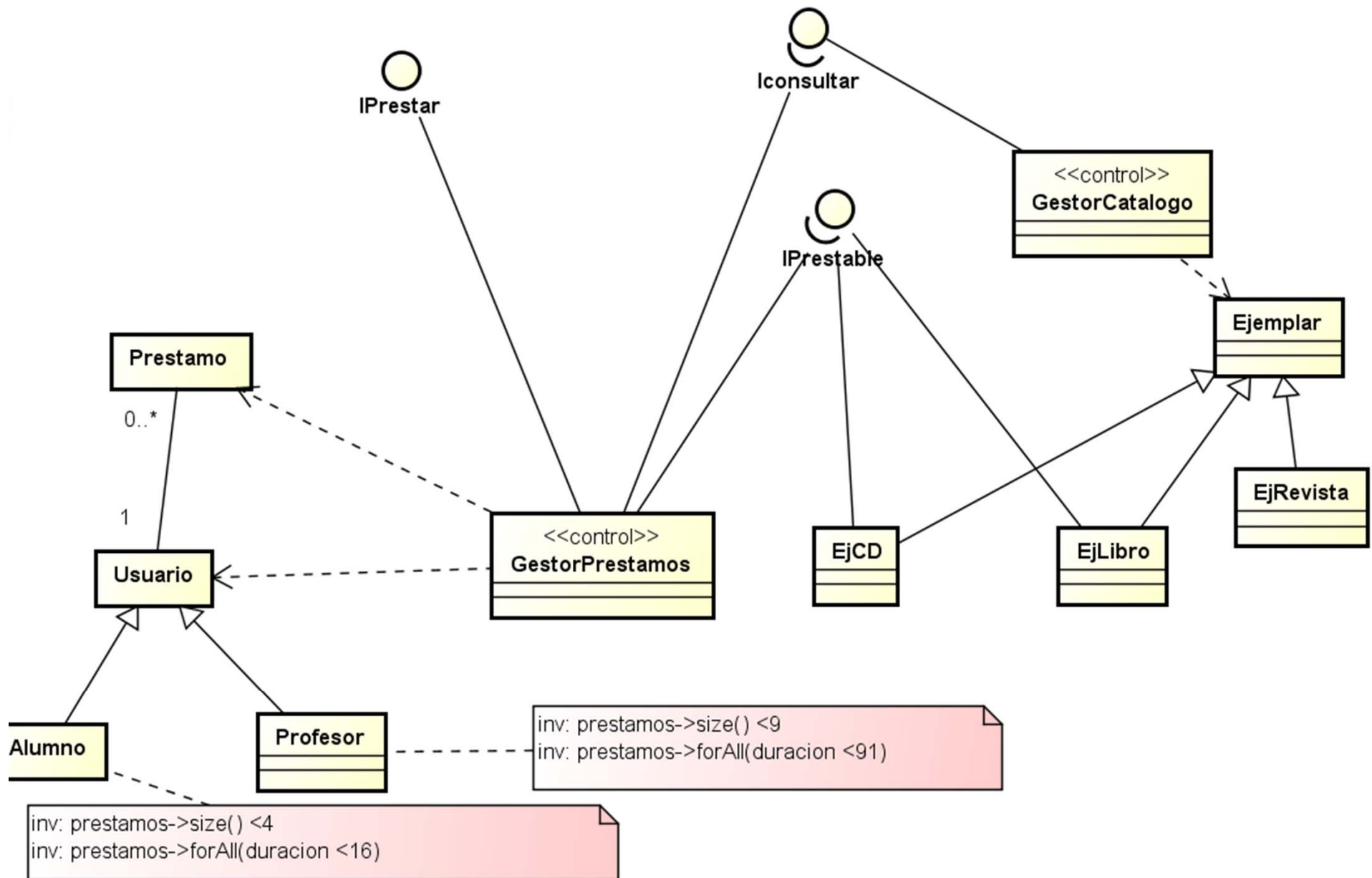
- IPrestar debe incluir las operaciones de prestar y devolver un ejemplar
- Iconconsultar incluirá búsqueda de ejemplares por título, por signatura, etc
- Caja negra:



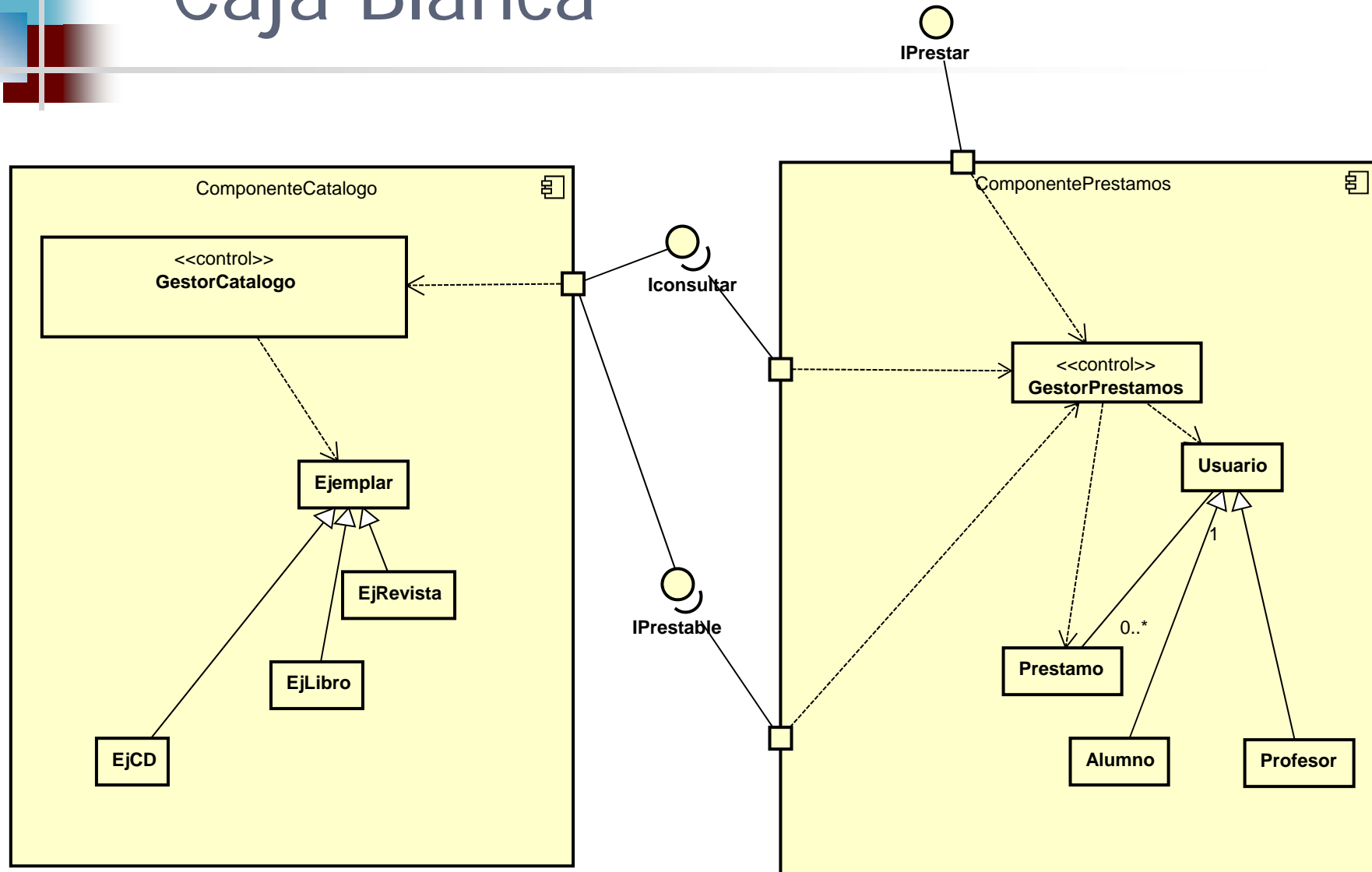


Encontrar más interfaces

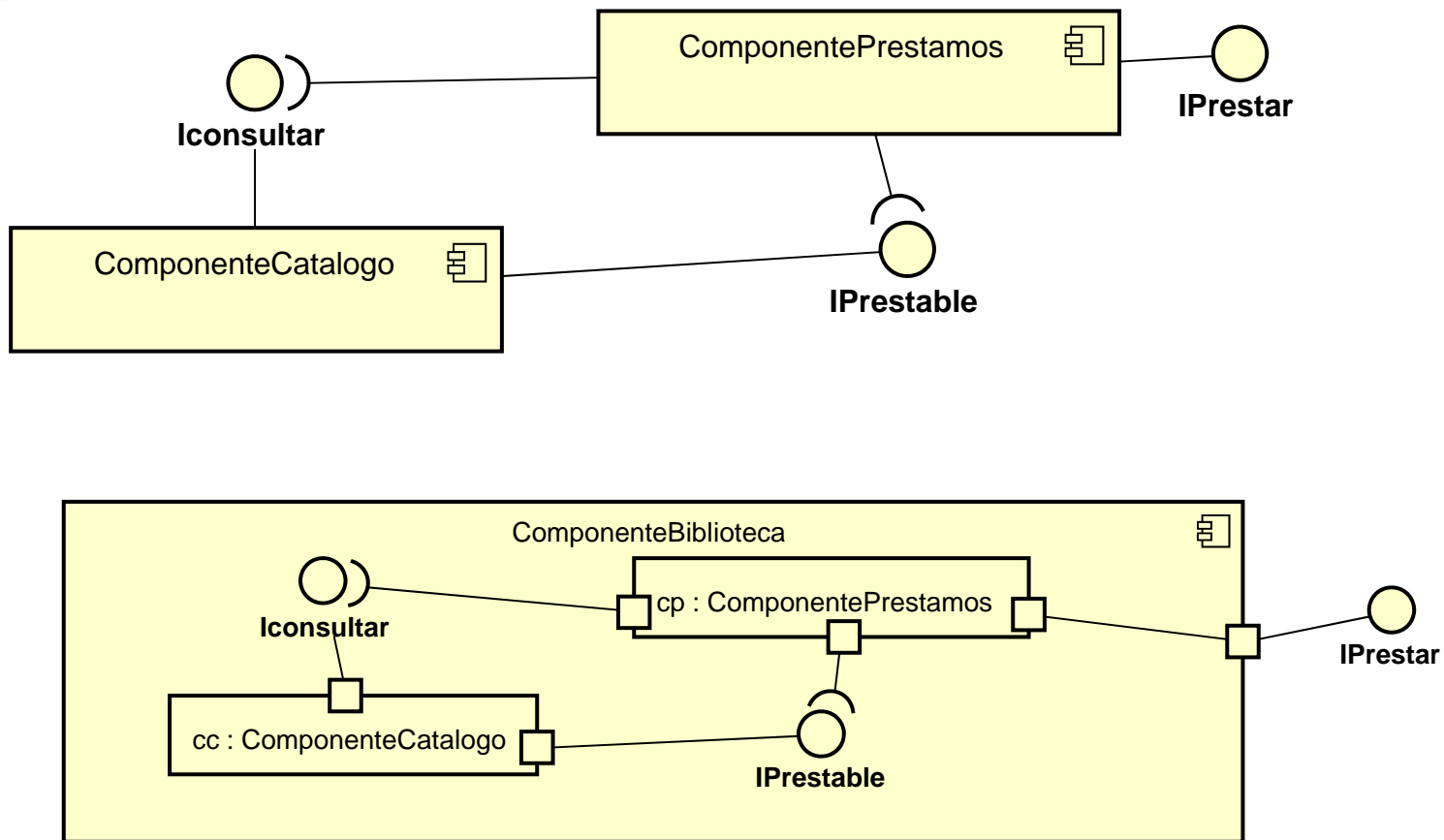
- Herencia e Interfaces!
- Estudiar cada asociación:
 - ¿La asociación tiene que ser a otra clase, o puede ser a una interfaz?
- Estudiar los mensaje enviados:
 - ¿El mensaje tiene que ser enviado a otra clase, o puede serlo a una interfaz?
- Buscar grupos de operaciones repetidas, que puedan ser útiles en otros lugares o tengan posibilidades de expansión futura → **PARA REUTILIZACIÓN**
- ¿Algún componente ya existe en la empresa? ¿Se puede adquirir? → **CON REUTILIZACIÓN**

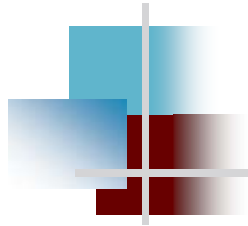


Caja Blanca



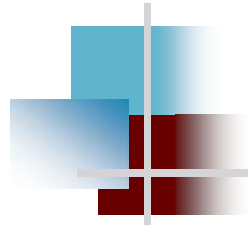
Caja Negra y Subsistema





3. Diseñar las Interfaces

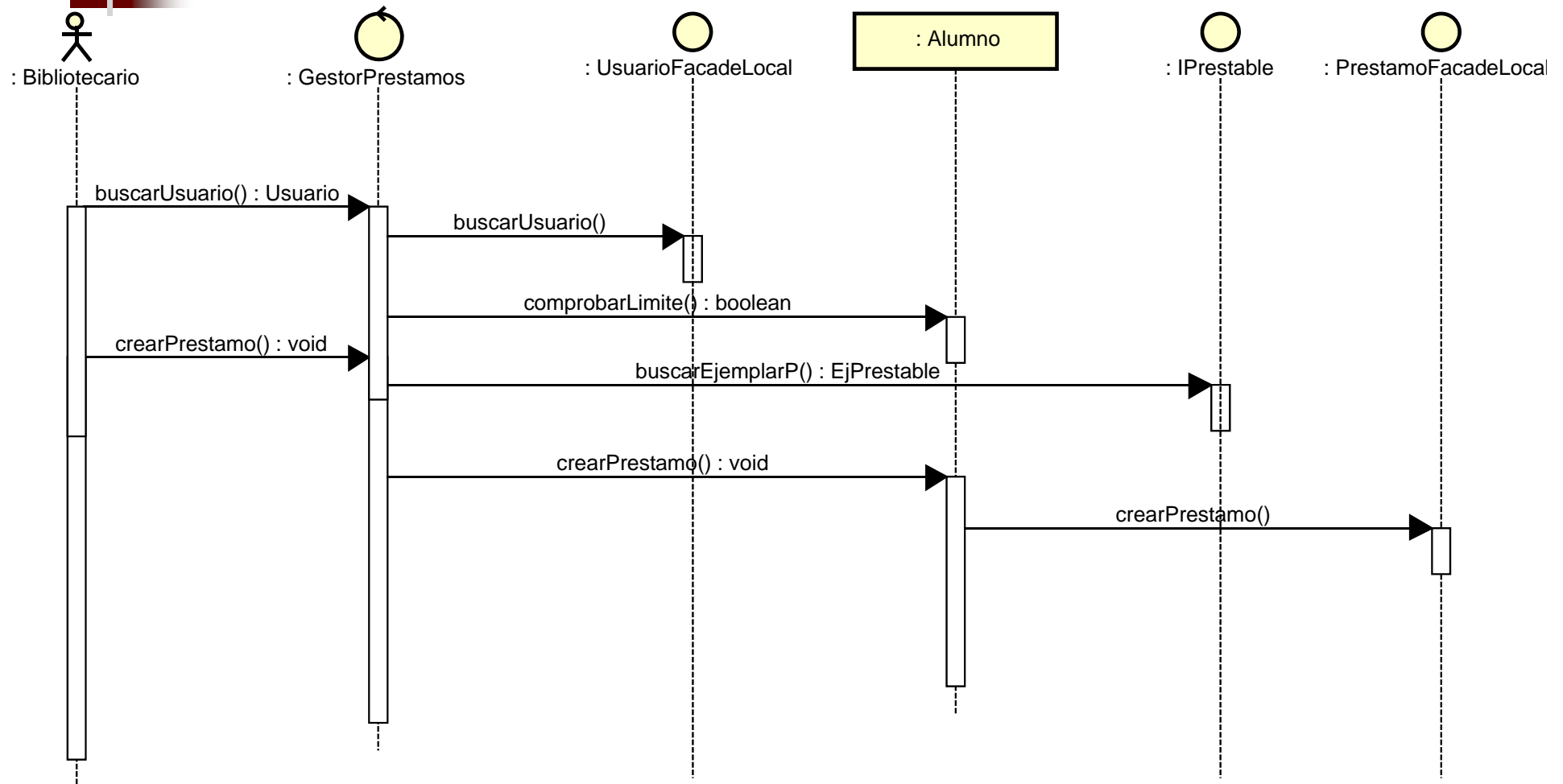
- Para cada interfaz del componente definir en detalle las Operaciones ofrecidas o requeridas por la interfaz:
 - Parámetros de entrada y salida
 - Pre y post-condiciones
 - Efectos de cada operación
 - Naturaleza de la Interfaz (mensaje asíncrono, RPC, Webservice)
- Para su definición se puede utilizar
 - UML+OCL
 - Lenguajes de programación
 - IDLs
- [Organizar en puertos]

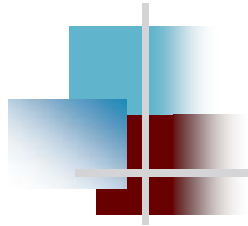


4 y 5. Verificar requisitos

- 4. Verificar los requisitos funcionales
 - Hacer el seguimiento de cada requisito, utilizando la estructura funcional
 - Utilizar una tabla de requisitos funcionales contra componentes del modelo estructural
- 5. Verificar los casos de uso
 - Analizar la estructura funcional propuesta, junto con los participantes, a través de los casos de uso.

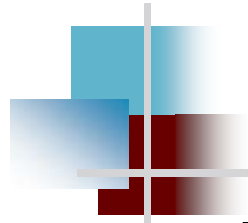
Detalle casos de uso





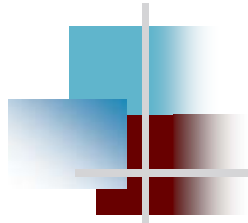
6 y 7. Análisis crítico

- 6. Análisis de Interacciones
 - Analizar la estructura propuesta en busca de interacciones excesivas (acoplamiento excesivo entre los componentes)
- 7. Análisis de Flexibilidad
 - Escenarios "what if ", ¿qué pasaría si...?



Algunos errores frecuentes

- Mala definición de interfaces
- Mala definición de responsabilidades
- Componentes de tipo utilidad modelados como componentes funcionales (listados, informes)
- Nivel inapropiado de detalle
- Número elevado de dependencias
- “God component” / “Manager”
 - 50% de las responsabilidades en menos del 25% de los componentes

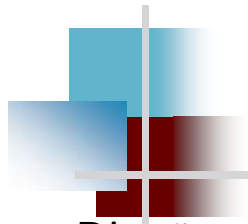


Lista de control

- ¿El modelo tiene un número razonable de componentes?
- ¿Todos los componentes tienen nombre, responsabilidades claras e interfaces claramente definidas?
- ¿Todas las interacciones entre los componentes ocurren a través de interfaces y conectores entre ellas
- ¿Los componentes tienen una alta cohesión?
- ¿Los componentes muestran un bajo acoplamiento?
- ¿Se ha validado la estructura propuesta contra los requisitos funcionales?
- ¿Ha considerado como se porta la arquitectura en escenarios hipotéticos de cambio?

Ejemplo: Videoclub



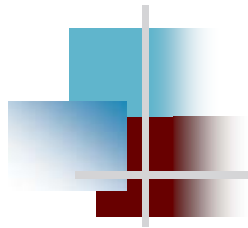


Ejemplo: Videoclub [on line]

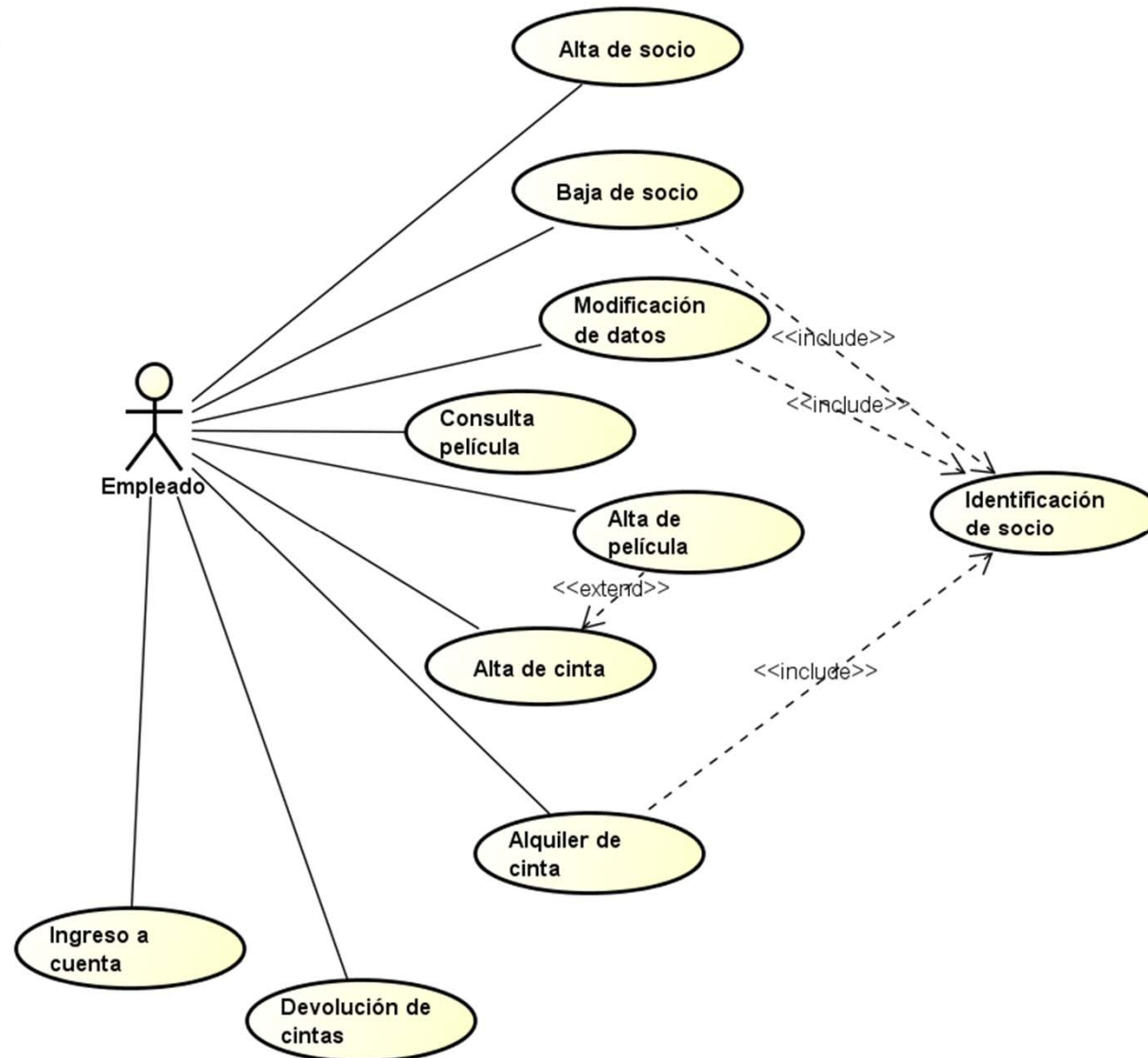
Diseñar un sistema basado en componentes a partir del ejemplo presentado en:

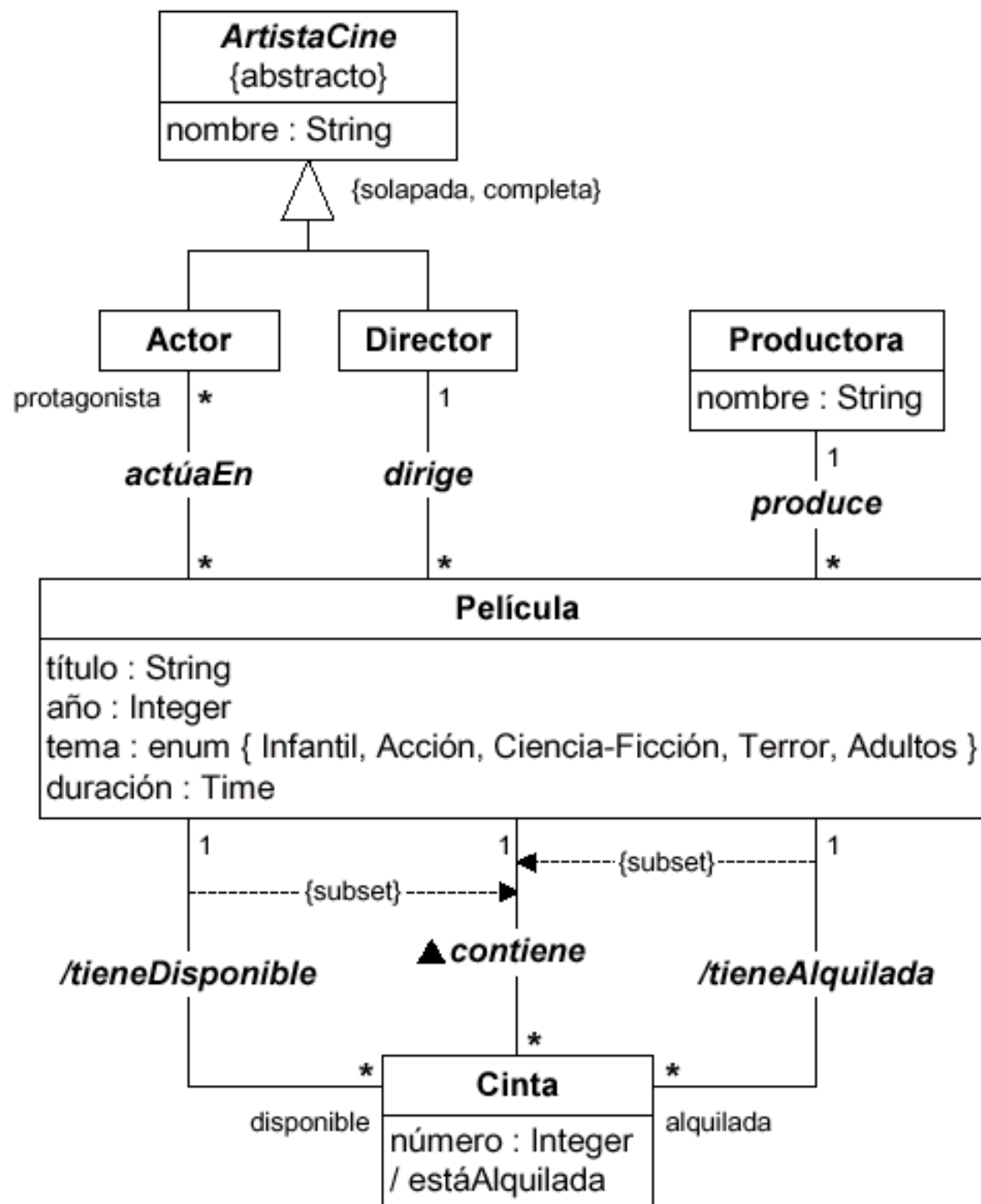
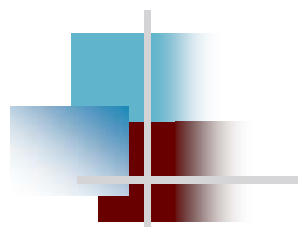
“Metodología para la Elicitación de Requisitos de Sistemas Software” y
“Metodología para el Análisis de Requisitos de Sistemas Software”, Amador Durán
Toro y Beatriz Bernárdez Jiménez. Universidad de Sevilla.

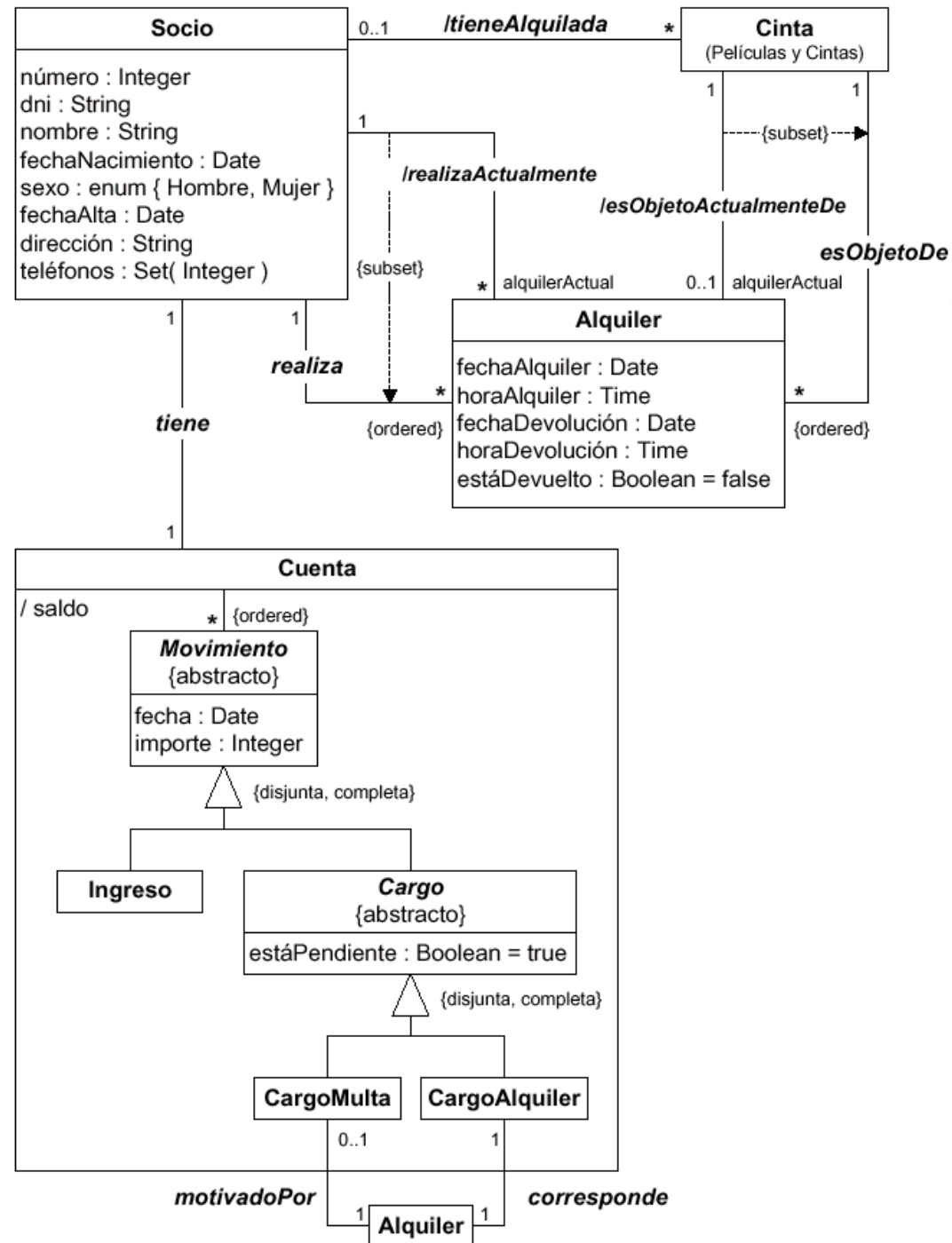
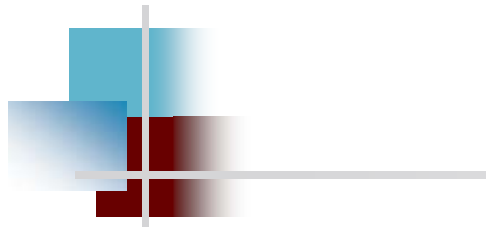
Se trata de un videoclub con películas (de cada una existen varios ejemplares cada una). Mantiene información sobre las películas, los socios y los alquileres. La forma de pago es con cargos y abonos en una cuenta.



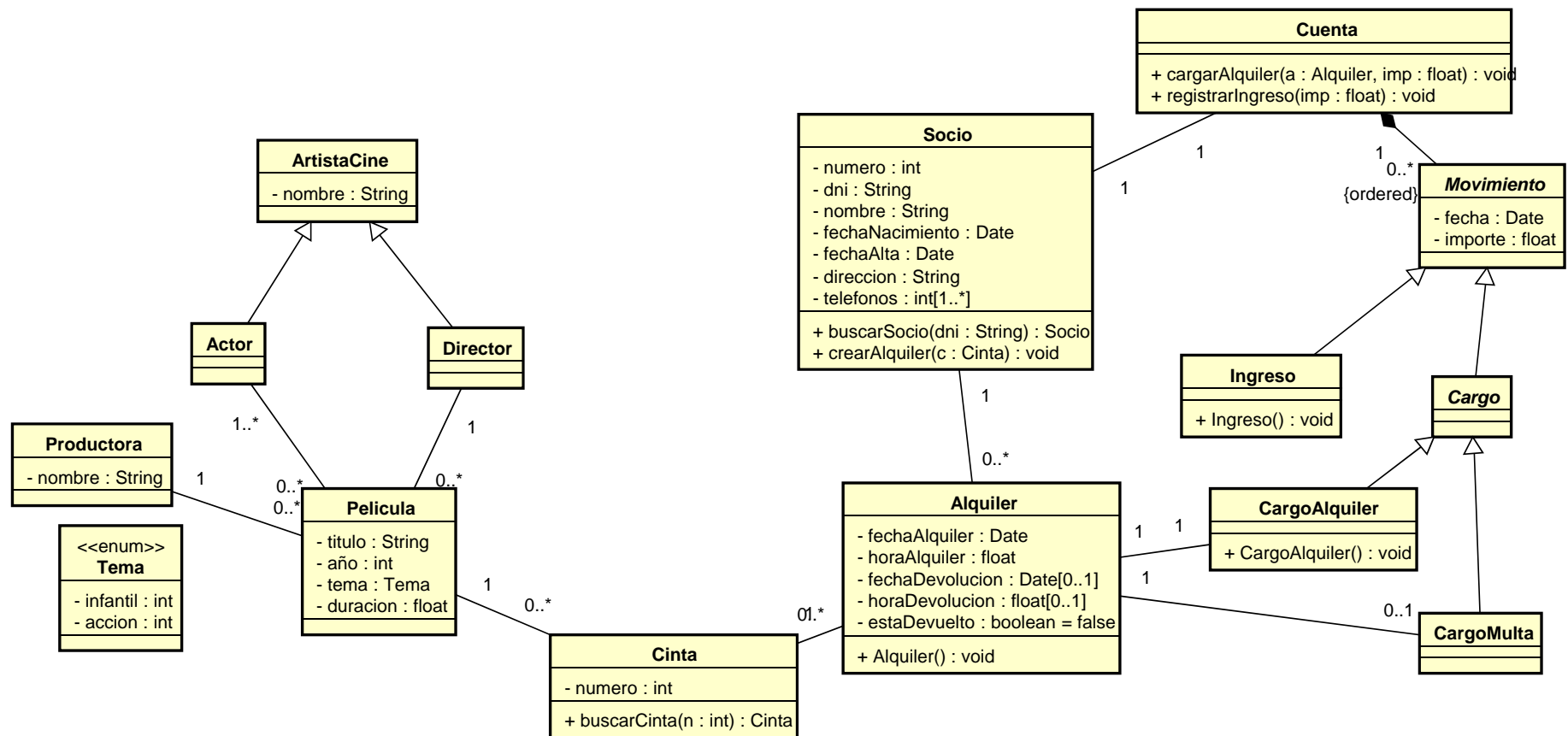
Algunos Casos de Uso





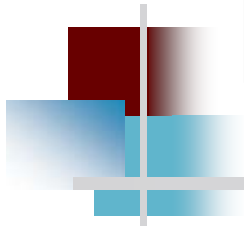


Modelo de Dominio



4.6.

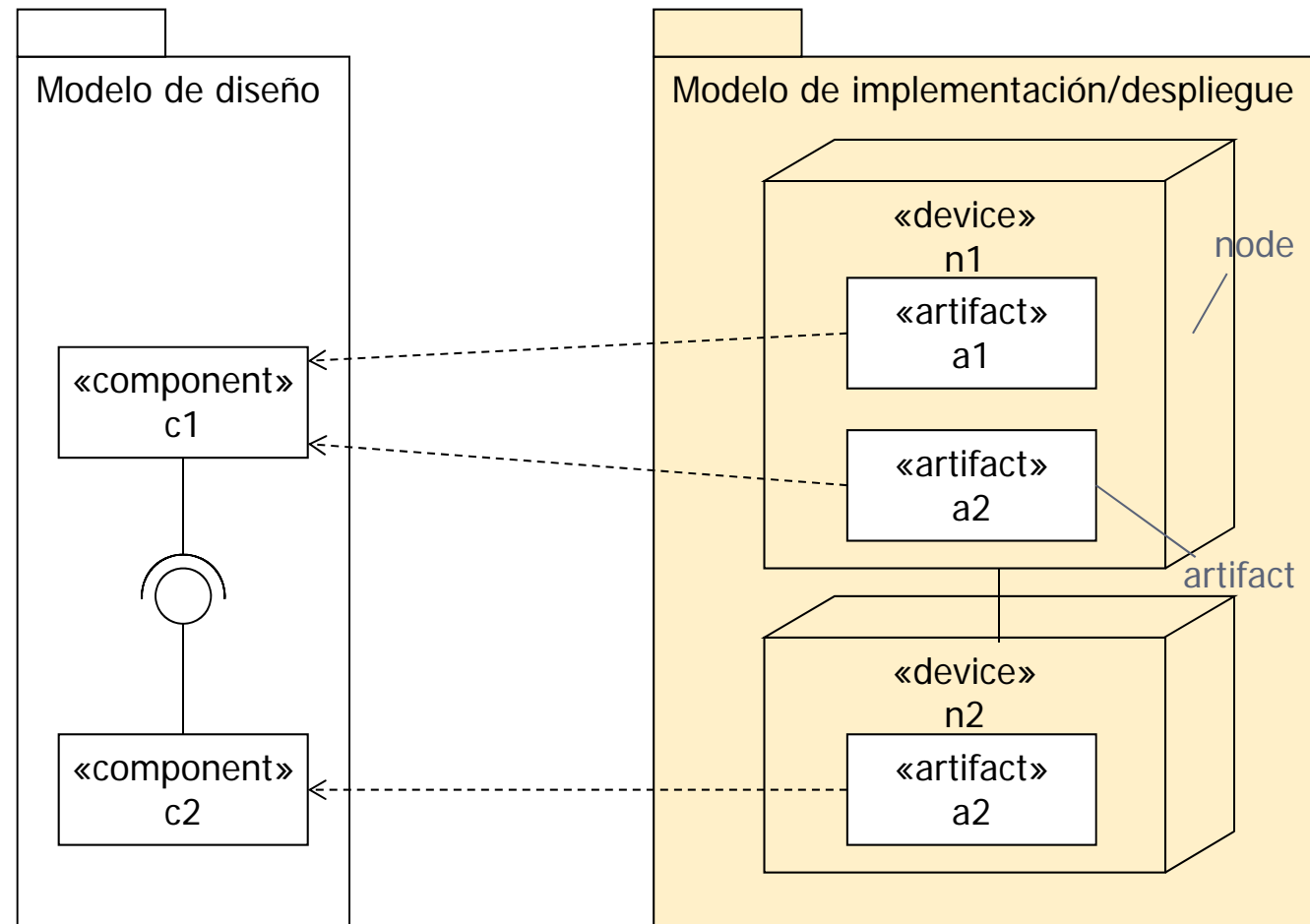
Despliegue de componentes



Implementación y despliegue

Los componentes se realizan con artefactos físicos que los implementan

El modelo de despliegue describe donde se encuentran físicamente



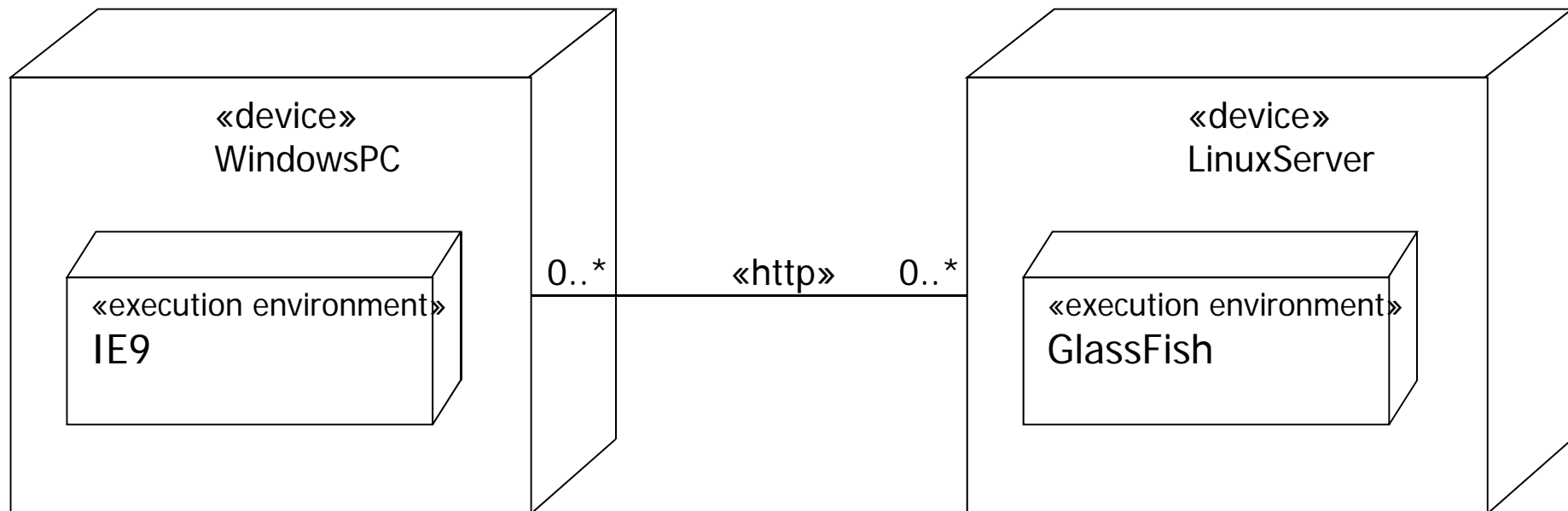


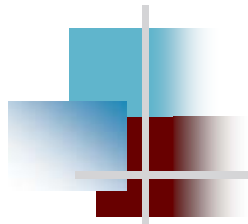
Modelo de despliegue

- Describe cómo se distribuye la funcionalidad en nodos físicos
 - Se modela la traducción de la arquitectura de software a la arquitectura del sistema físico
- Es el modelo del sistema como artefactos desplegados en los nodos o recursos computacionales
 - Los nodos tienen relaciones que representan los métodos de comunicación entre ellos, por ejemplo http, iiop, netbios
 - Los artefactos representan software, por ejemplo, un .JAR, .class o .exe
- Pasos del Diseño de la arquitectura física:
 - Determinar nodos y conexiones
 - Despliegue de artefactos

UML: diagramas de Despliegue

- «device» y «execution environment» son estereotipos estándar



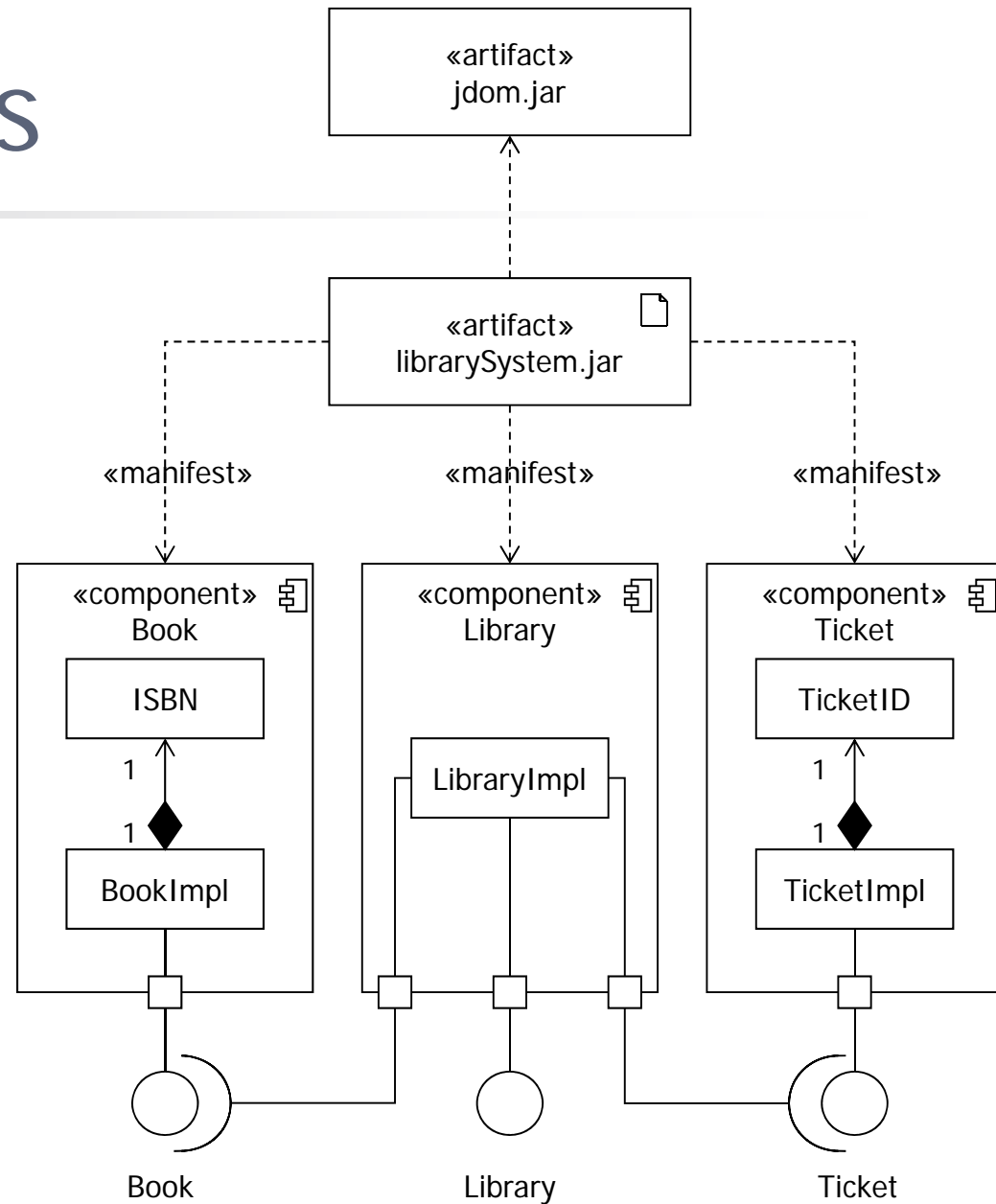


Artefactos

- Un artefacto representa un elemento concreto, como un archivo
 - Puede ser desplegado en los nodos
 - Instancias de Artefactos se despliegan en instancias de nodos
- Un artefacto puede “manifestar” uno o más componentes
 - El artefacto representa la manifestación física de los componentes (por ejemplo, un archivo JAR con varios EJBs)

Relaciones

- Con los componentes que realiza
- Con otros artefactos:
 - Dependencia
 - Composición



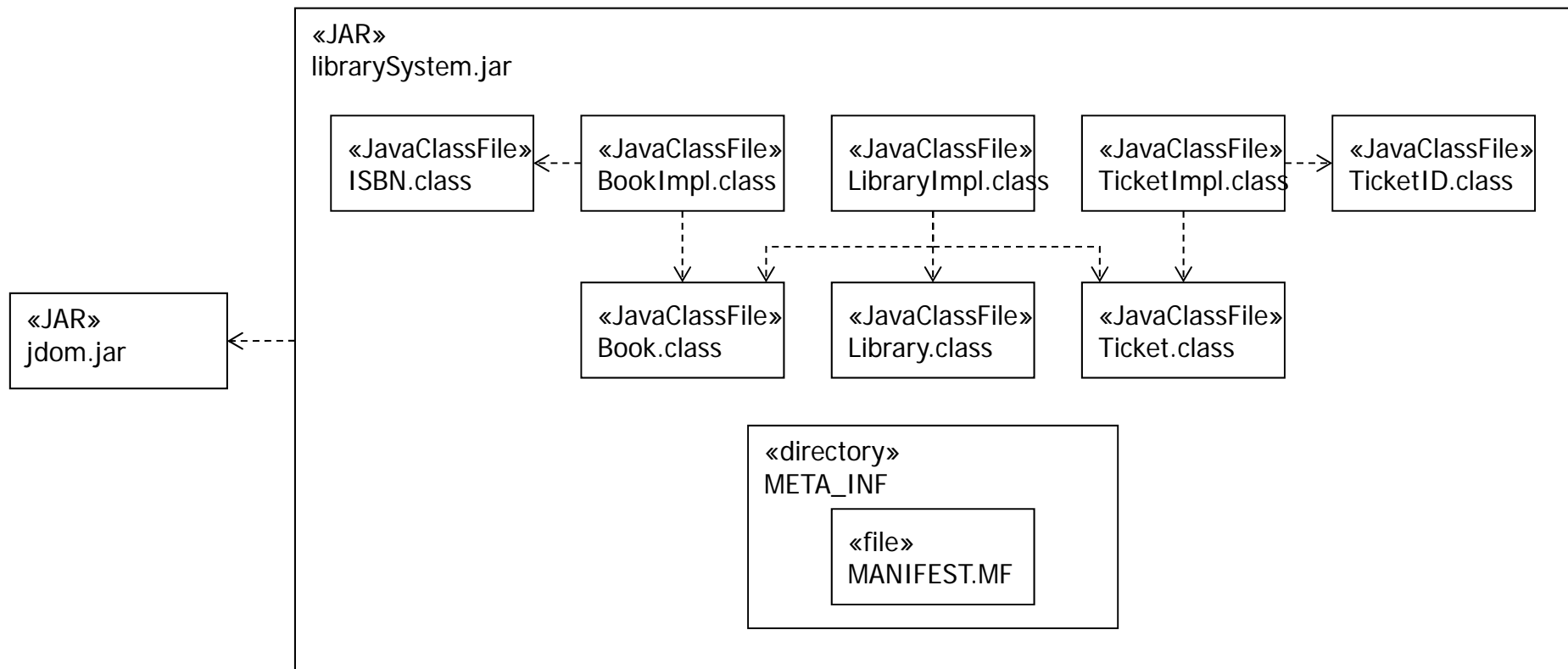


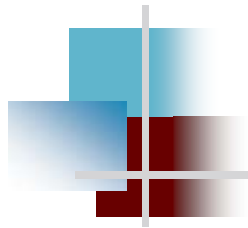
Estereotipos estándar UML

- «file»
- «deployment spec» persistence.xml
- «document» .DOC, .TXT, ...
- «executable» .EXE, .class, ...
- «library» .DLL o .jar
- «script» .js
- «source» .java

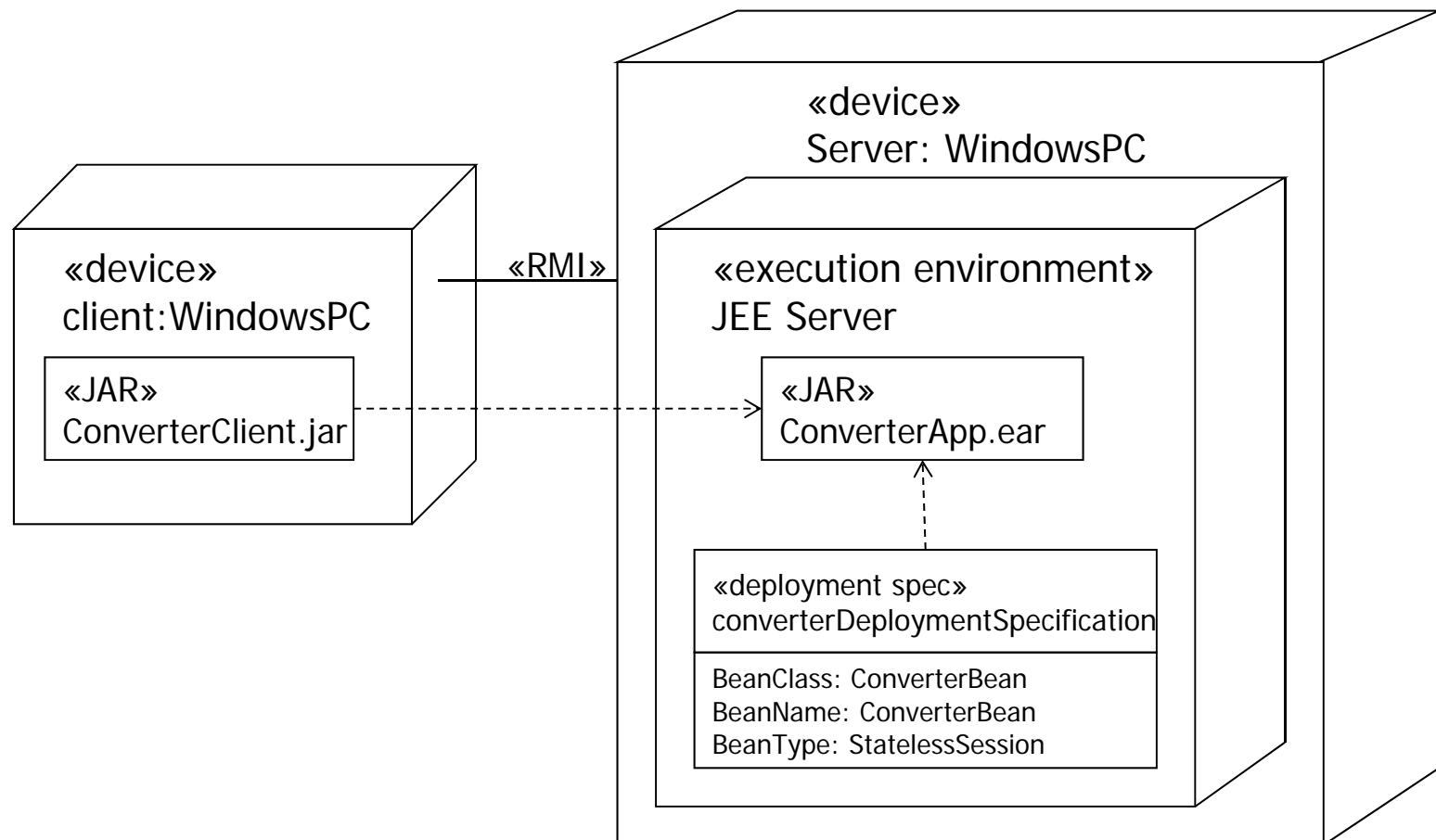
UML Java Profile

- Además de puede utilizar un perfil específico (conjunto de estereotipos para Java)



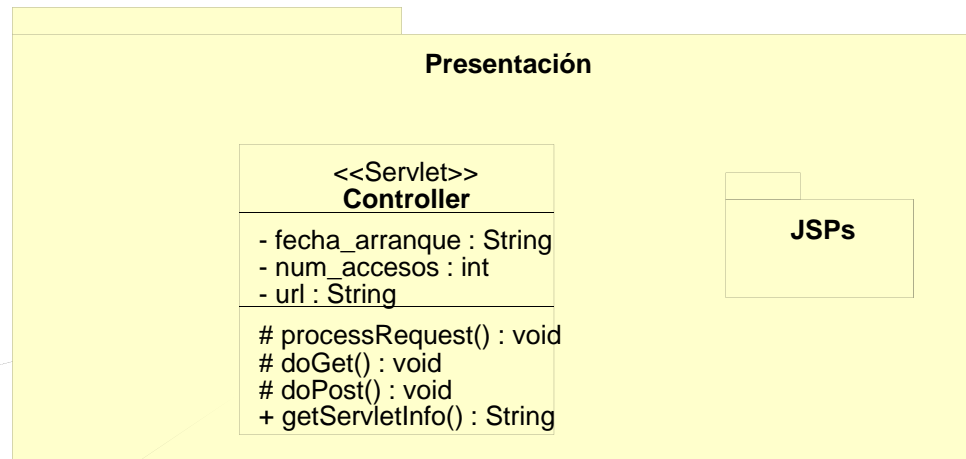
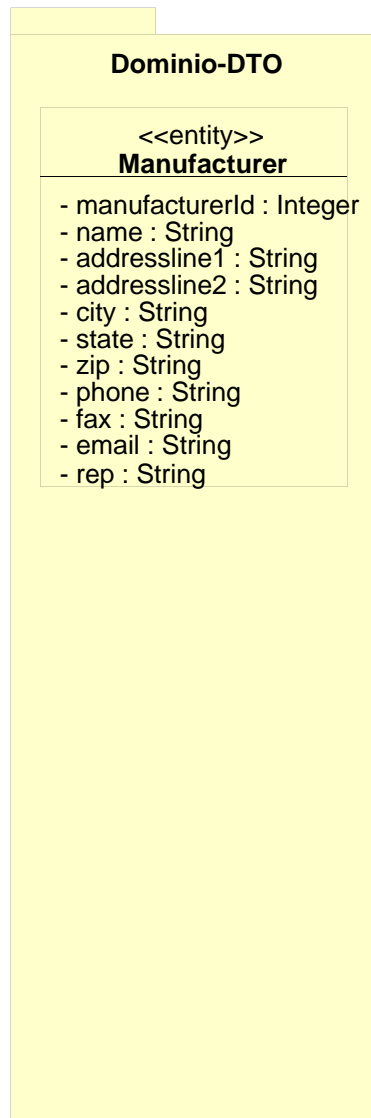


Artefactos y nodos

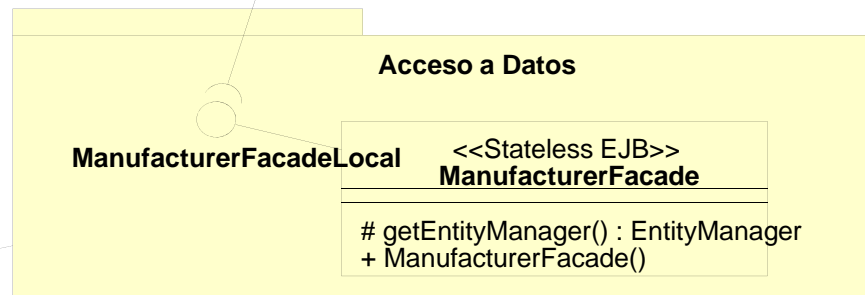
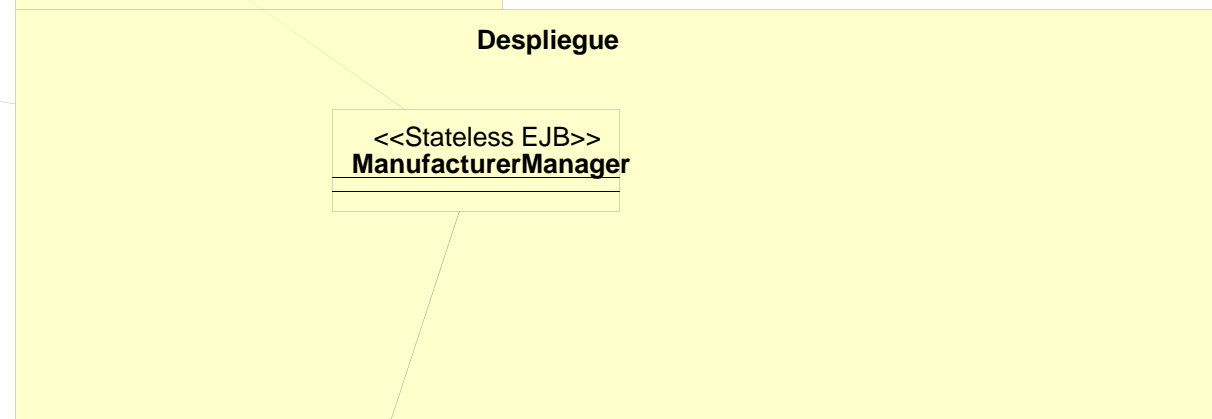




Ejemplo simple



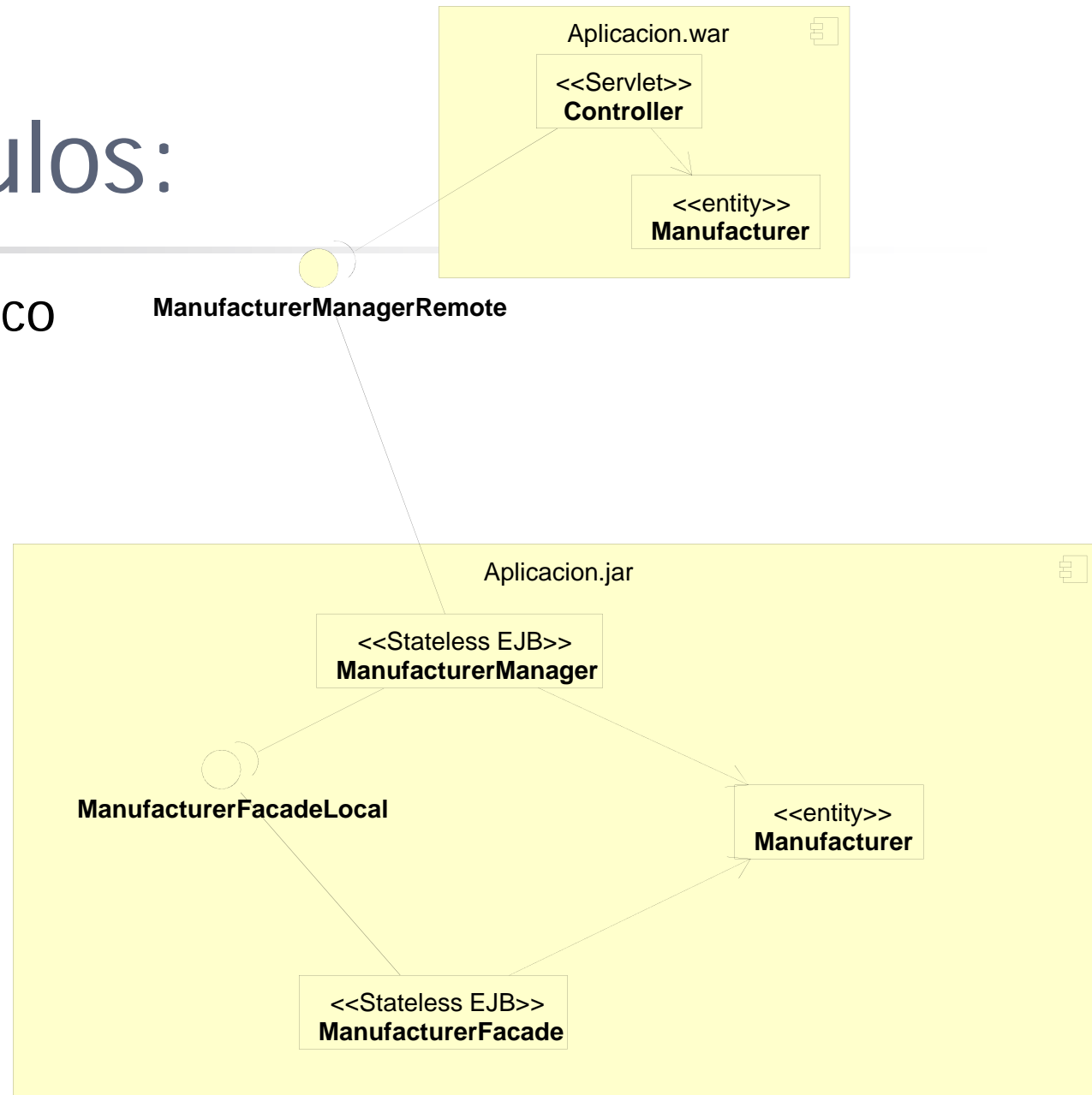
ManufacturerManagerRemote



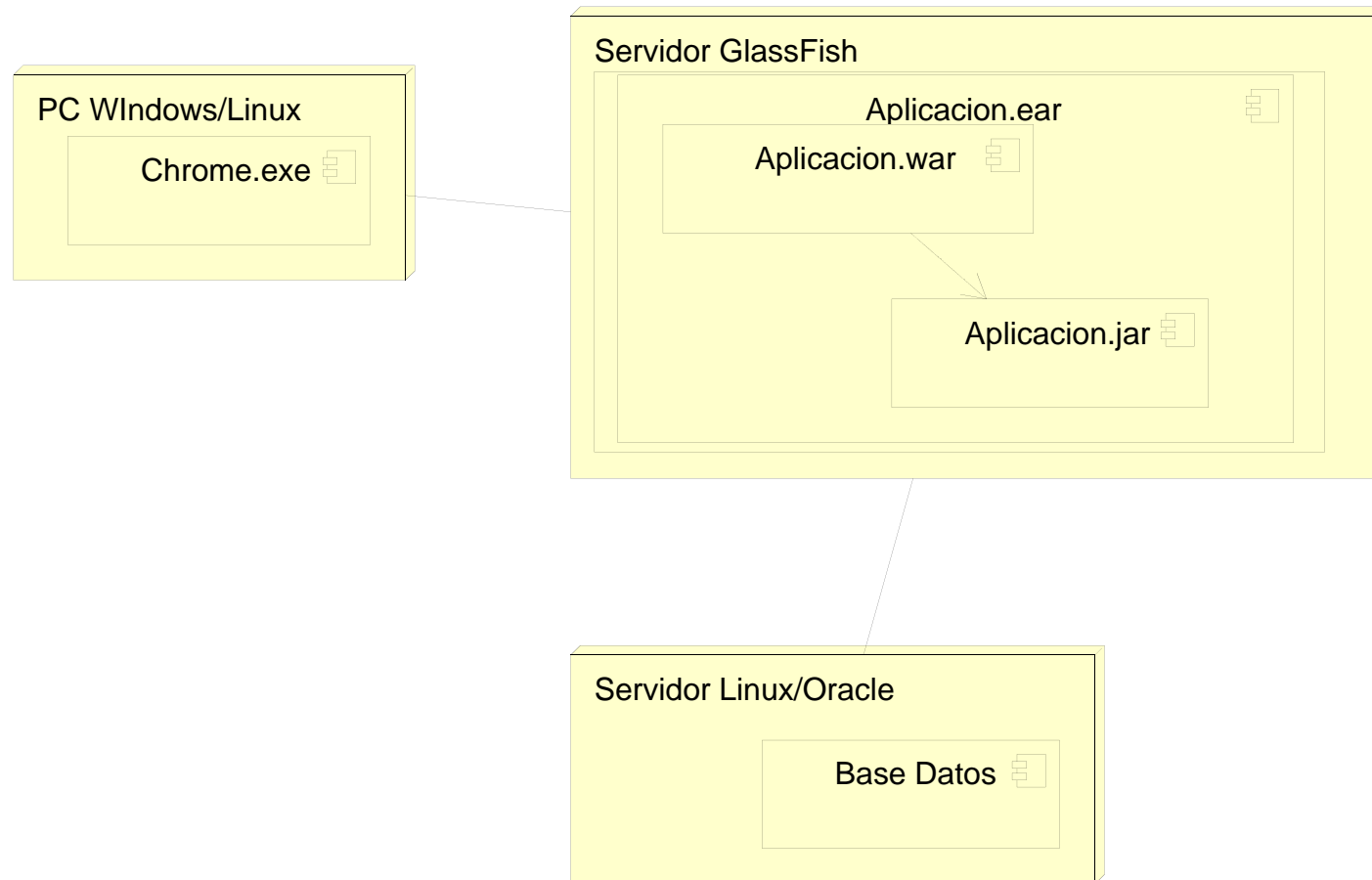
Módulos:

- Ejemplo típico con dos proyectos:

- .war
- .jar

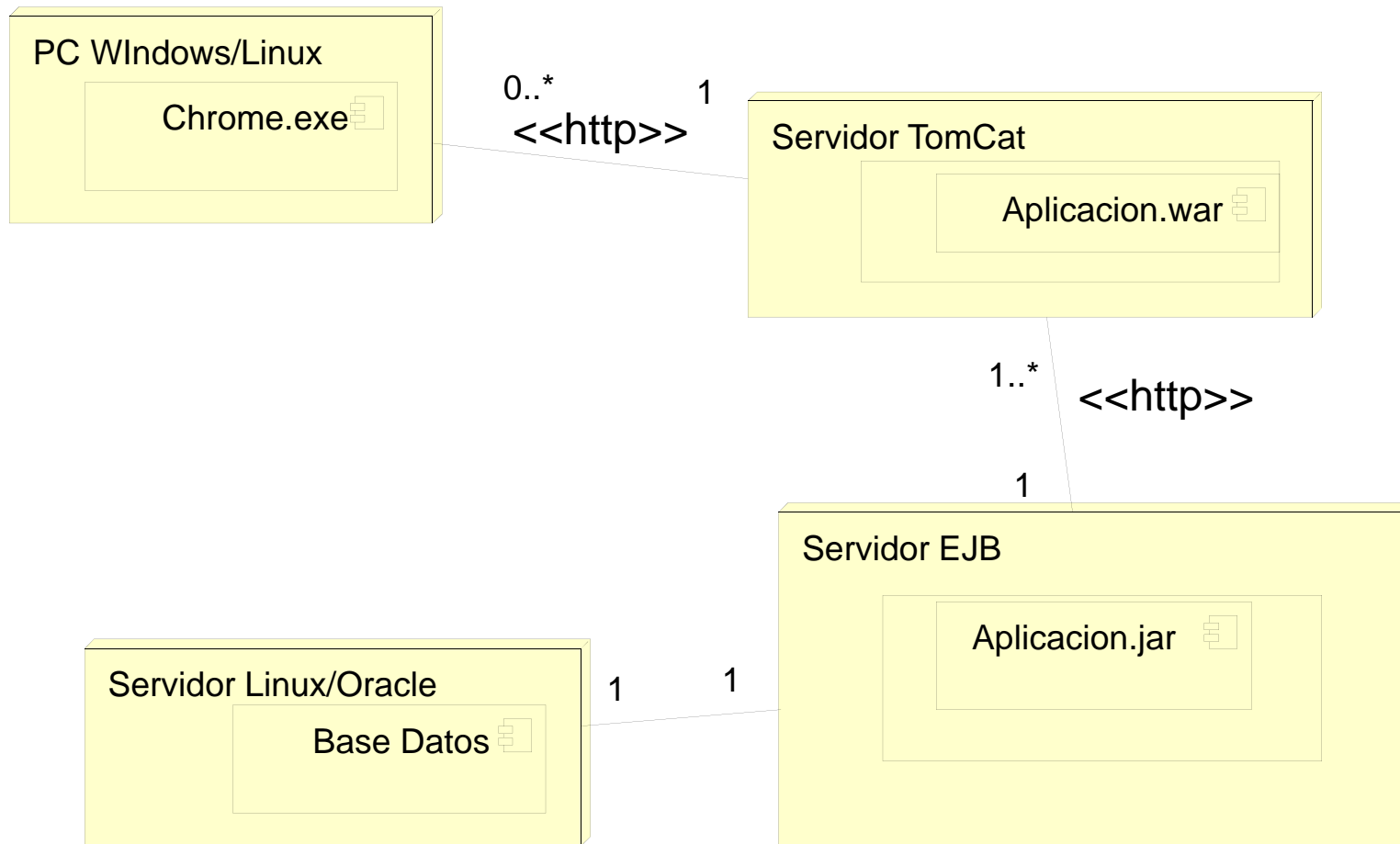


Un servidor (y servidor BD)



- (en este caso, mejor todas las interfaces locales)

Dos servidores: Servidor Web y Servidor JEE





Despliegue y Diseño: Recordatorio

- Cuando se usa más de un servidor, hay que definir interfaces remotas
- Eso influye en la forma de diseñar el Acceso a Datos:
 - perdemos la referencia directa a la instancia
 - No hay acceso directo a la clases entity (hay que colocarlas en una biblioteca compartida)
- El rendimiento disminuye con interfaces remotas y con los EJB Stateful