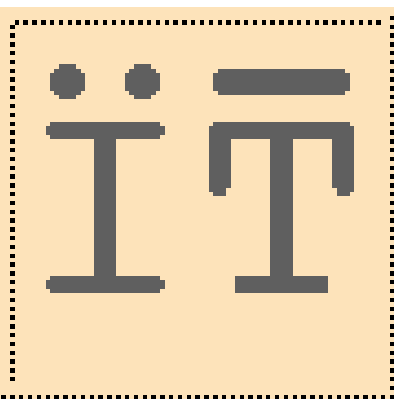


Introducción al modelo de componentes de CORBA: motivación y visión general

Simon Pickin

Departamento de Ingeniería Telemática

Universidad Carlos III de Madrid

simon.pickin@uc3m.es

Aplicaciones corporativas

- Un requisito fundamental:
  - gestión eficaz del almacenamiento y acceso a datos
- Algunos aspectos a tratar
  - SGBD
  - conectividad a la BD
    - BD distintas
    - BD múltiples
    - sistemas heredados (*legacy systems*)
  - representación de datos
    - correspondencia entre la representación del programa y la de la BD
  - integridad de los datos:
    - control de accesos concurrentes
    - monitores de transacciones
  - acceso rápido:
    - cachés de datos
  - control de acceso:
    - autenticación

Modelos de componentes (I)

- ¿Qué es un componente?: un módulo de software:
  - orientado al desarrollo de aplicaciones por ensamblaje de módulos existentes
  - que facilita la división del trabajo (responsabilidades claras)
  - que se puede escoger “de la estantería”, listo para su empleo (COTS: *Components “Off-The-Shelf”*): unidad de reuso
  - que se compila y despliega de manera independiente: unidad de despliegue
- Podemos requerir también:
  - que pueda formar parte de una aplicación distribuida
  - que sea de uso flexible $\Rightarrow$ múltiples interfaces
  - que se comunique de manera flexible
    - comunicación síncrona por invocación de métodos
    - comunicación asíncrona por canales (eventos)

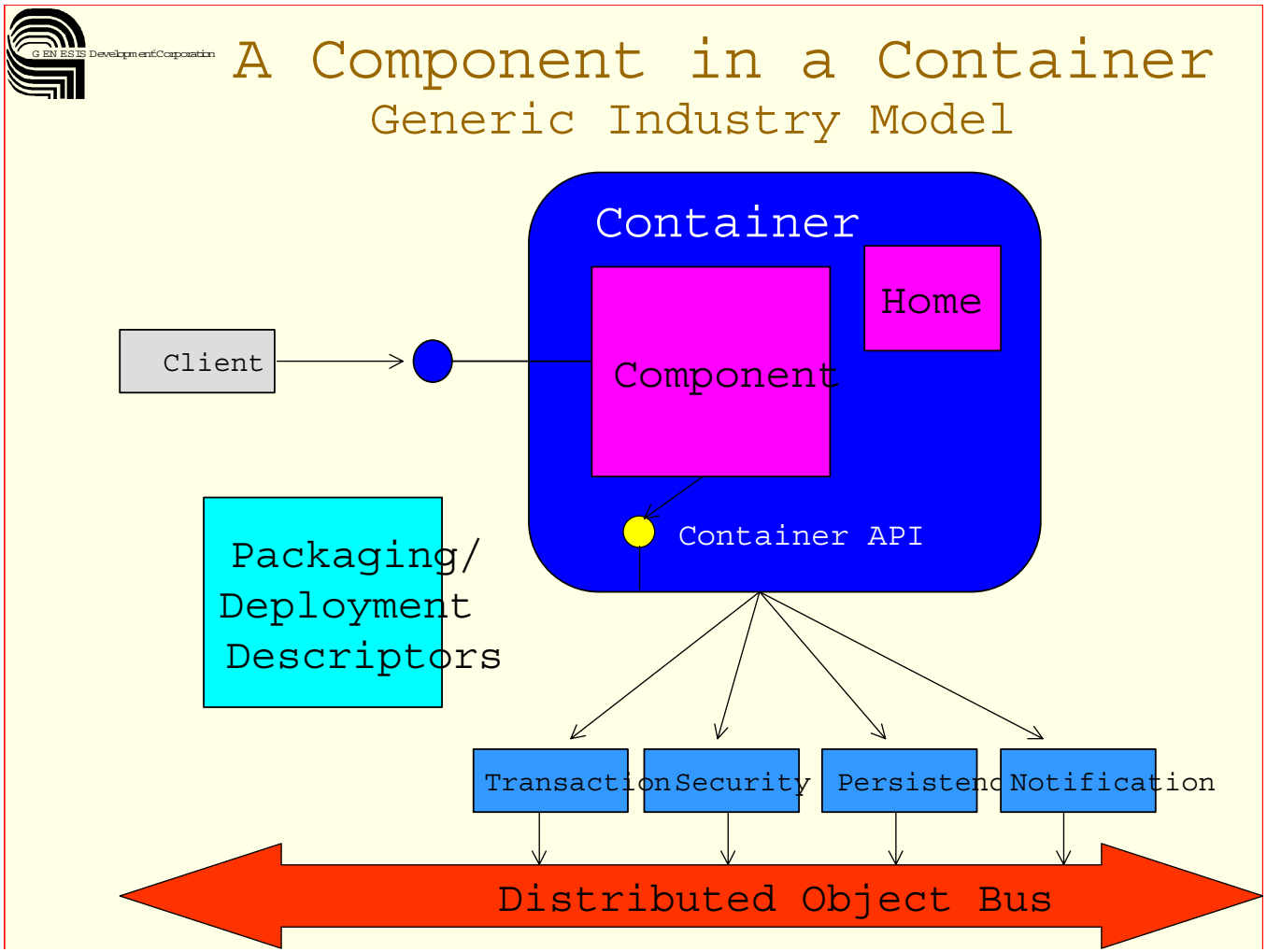
Modelos de componentes (II)

- ¿Qué incluye un modelo de componentes?
  - una noción de componente individual
  - una definición de cómo ensamblar componentes
    - conectando interfaces compatibles
  - Una definición de un entorno de componentes
    - es decir, un entorno de despliegue y ejecución de componentes
- ¿Qué proporciona un modelo de componentes?
  - diseño y desarrollo por ensamblaje: reuso
  - visión clara de la arquitectura de una aplicación
  - separación de los aspectos funcionales y no funcionales
- No cumplen las condiciones anteriores:
  - modelos basados en objetos, p.e. RMI,...
  - modelos basados en servicios, p.e. CORBA 2, Jini,...

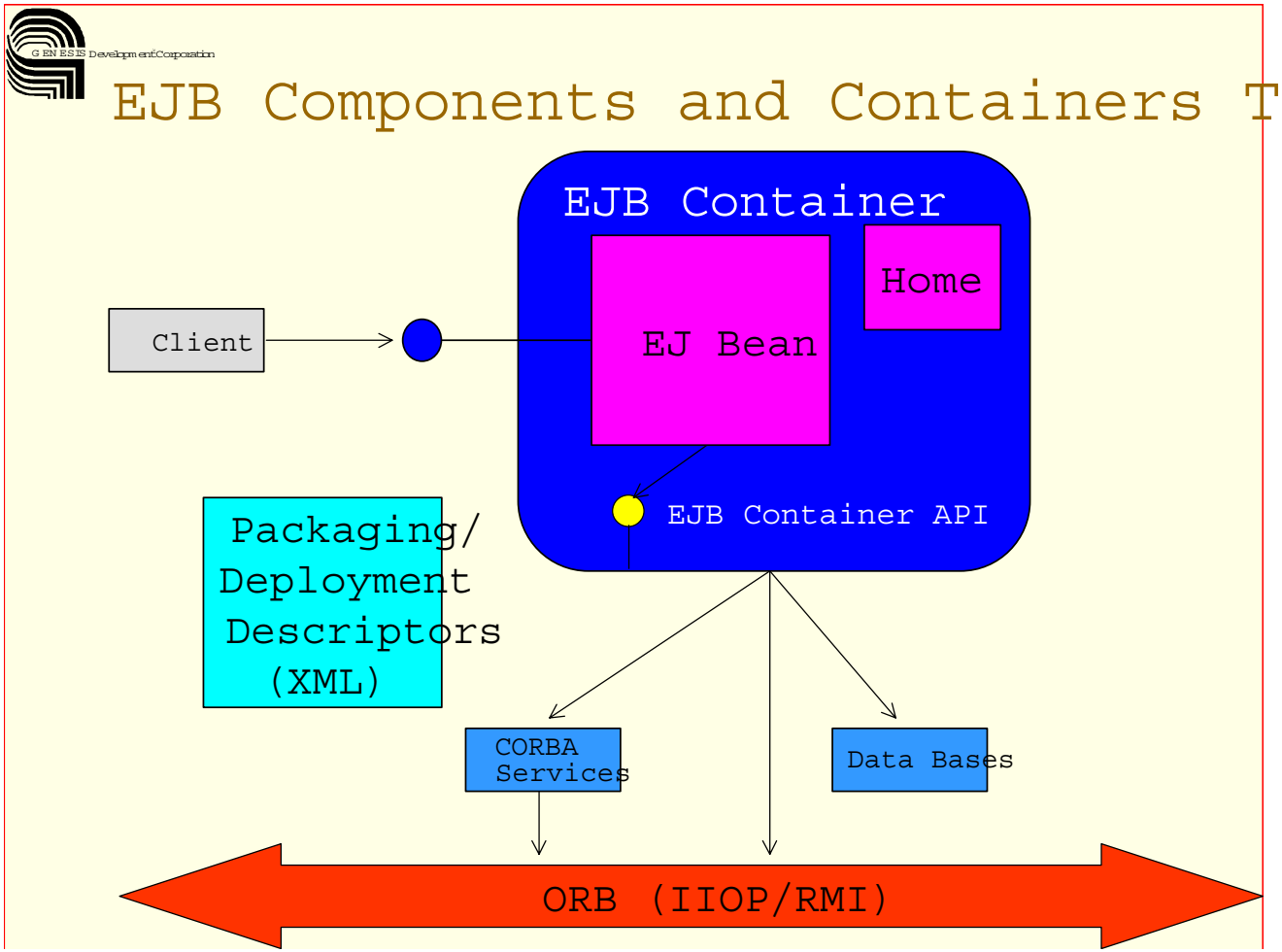
Entornos de componentes distribuidos (I)

- ¿Qué es un entorno de componentes distribuidos?
  - un entorno concebido para el despliegue y ejecución de aplicaciones distribuidas basadas en componentes
- ¿Qué conlleva un entorno de componentes distribuidos?
  - la separación de los aspectos funcionales y no funcionales
  - la gestión y soporte *implícitos* por el entorno de ejecución (vía perfiles estándar) de los aspectos no funcionales tales como:
    - seguridad (autenticación, autorización,...)
    - transacciones (definición declarativa o vía API)
    - control de concurrencia
    - persistencia (gestionada por el entorno o vía API)
    - gestión de ciclo de vida
    - nombrado (*naming*), *trading*, búsqueda de componentes
    - activación / desactivación
    - protocolos de comunicación
    - administración de componentes
  - soporte para el despliegue

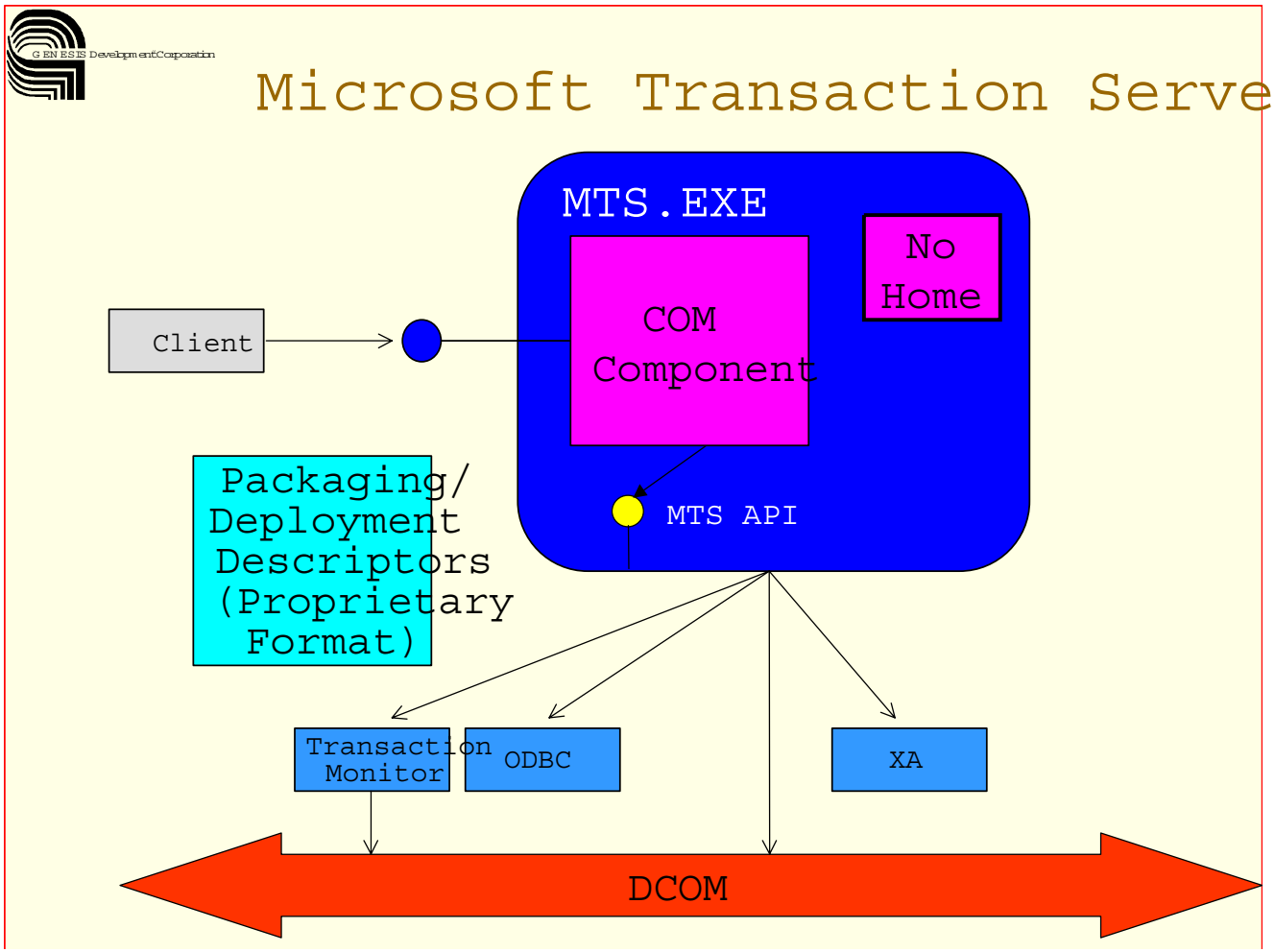
Entornos de componentes distribuidos (II)



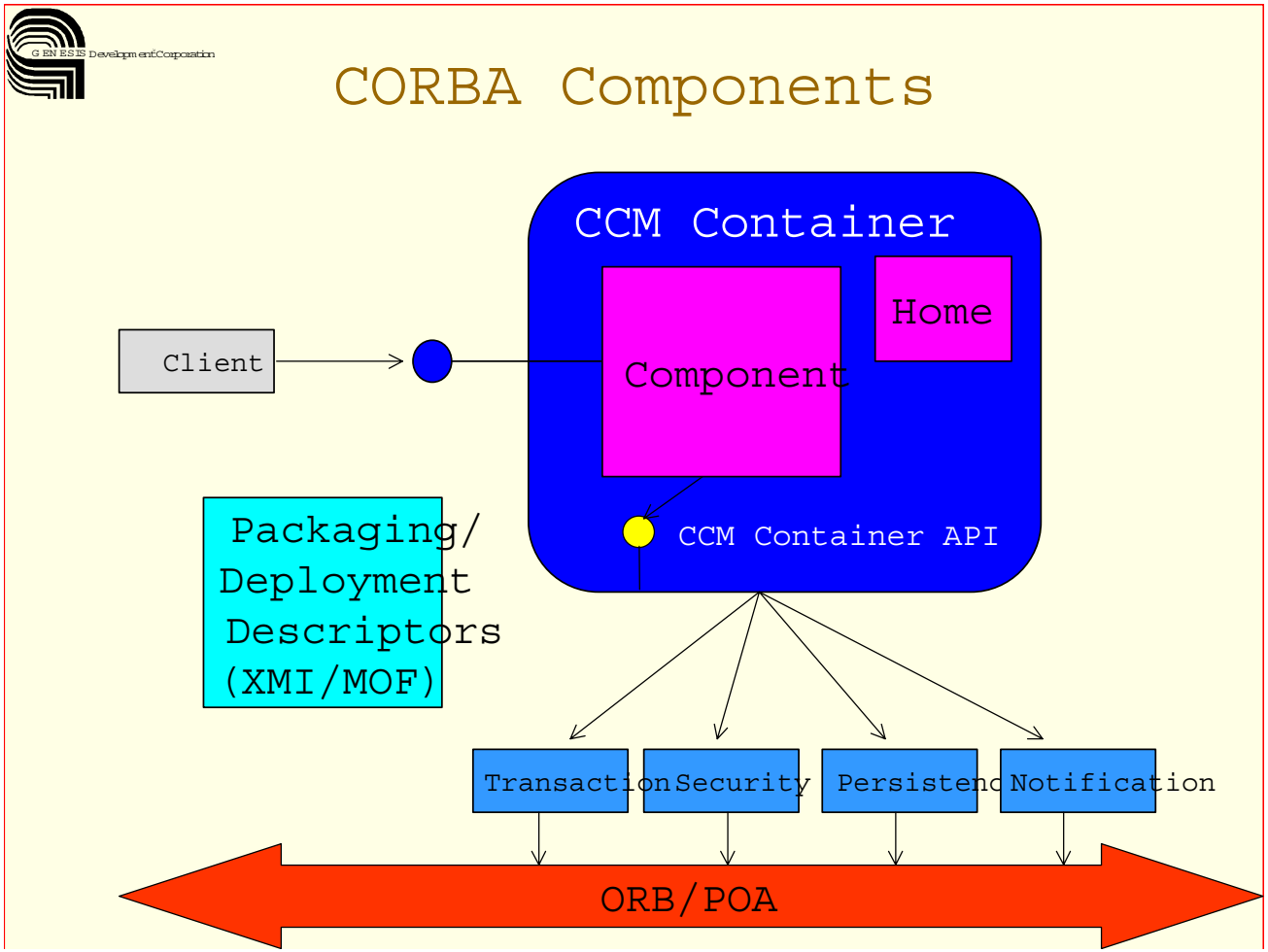
Entornos de componentes distribuidos (III)



Entornos de componentes distribuidos (IV)



Entornos de componentes distribuidos (V)



El modelo de componentes de CORBA 3

- CORBA 3: estándar del OMG de junio de 2002
  - salto significativo con respecto a CORBA 2
  - ¿nuevo impulso para CORBA?
  - novedad más importante: CCM (modelo de componentes)
  - otras mejoras: calidad de servicio, integración Internet
- Fuentes de inspiración para CCM
  - objetos computacionales de TINA (y de ODP)
  - sobre todo, EJB (*Enterprise Java Beans*) de Sun
  - Microsoft Transaction Server
- Mercado inicial previsto para CCM (y también EJB)
  - aplicaciones corporativas (*enterprise applications*)

Desde CORBA 2 . . .

- Un modelo de objetos distribuidos
  - heterogeneidad: OMG *Interface Definition Language*
  - portabilidad: mapeo de IDL a distintos lenguajes estandarizado
  - interoperabilidad: GIOP / IIOP
  - distintos modelos de invocación: SII, DII y AMI (CORBA 2.4)
  - *middleware*: ORB, POA, etc.; distintos perfiles
- No hay facilidad de empaquetado y despliegue
- Propiedades no funcionales se programan explícitamente
  - ciclo de vida, (des)activación, nombres, *trading*, notificación, persistencia, transacciones, seguridad, tiempo real, tolerancia a fallos, ...
- Ausencia de una visión de la arquitectura de software

. . . hasta el CCM de CORBA 3

- Un modelo distribuido orientado a componentes
  - una arquitectura para definir componentes y sus relaciones
    - tanto en el lado cliente como en el lado servidor
  - una tecnología de empaquetado para el despliegue de ejecutables multi-lenguaje binarios
  - Una noción de contenedor para inyectar servicios de ciclo de vida, (des)activación, seguridad, transacciones, persistencia y eventos
  - interoperabilidad con *Enterprise Java Beans* (EJB)
- El primer estándar de componentes abierto
  - CCM: multi-lenguaje, multi-SO, multi-ORB, multi-vendedor, etc.
  - modelo EJB: restricción a Java
  - modelo .NET: restricción a SO Microsoft

CCM comparado con EJB, COM y .NET

- Como *Enterprise Java Beans* (EJB) de SUN Microsystems
  - componentes CORBA creados y gestionados por *homes*
  - ejecutan en contenedores que gestionan los servicios de sistema de manera transparente
  - alojados en servidores de componentes de aplicación
- Como *Component Object Model* (COM) de Microsoft
  - tienen varios interfaces de entrada y de salida
    - tanto operaciones síncronas como eventos asíncronos
  - posibilidades de navegación e introspección
- Como .NET de Microsoft
  - pueden escribirse en distintos lenguajes de programación
  - pueden empaquetarse para distribución

Introducción al modelo de componentes de CORBA: CCM (*CORBA Component Model*)

Simon Pickin

Departamento de Ingeniería Telemática

Universidad Carlos III de Madrid

simon.pickin@uc3m.es

Indice

- Introducción
- Modelo abstracto
- Modelo de programación
- Modelo de ejecución
- Modelo de despliegue
- Relación / comparación CCM - EJB

Los cuatro modelos del CCM

- Modelo abstracto
  - uso de IDL3 (*Interface Definition Language* de CORBA 3)
- Modelo de programación
  - uso de CIDL (*Component Implementation Definition Language*)
  - uso del CIF (*Component Implementation Framework*)
  - uso de IDL2 (*Interface Definition Language* de CORBA 2)
- Modelo de ejecución
  - contenedores
  - servidores de aplicaciones
- Modelo de despliegue
  - uso de OSD (*Open Software Description*)

Indice

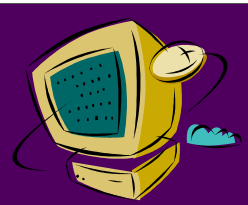
- Introducción
- Modelo abstracto
- Modelo de programación
- Modelo de ejecución
- Modelo de despliegue
- Relación / comparación CCM - EJB

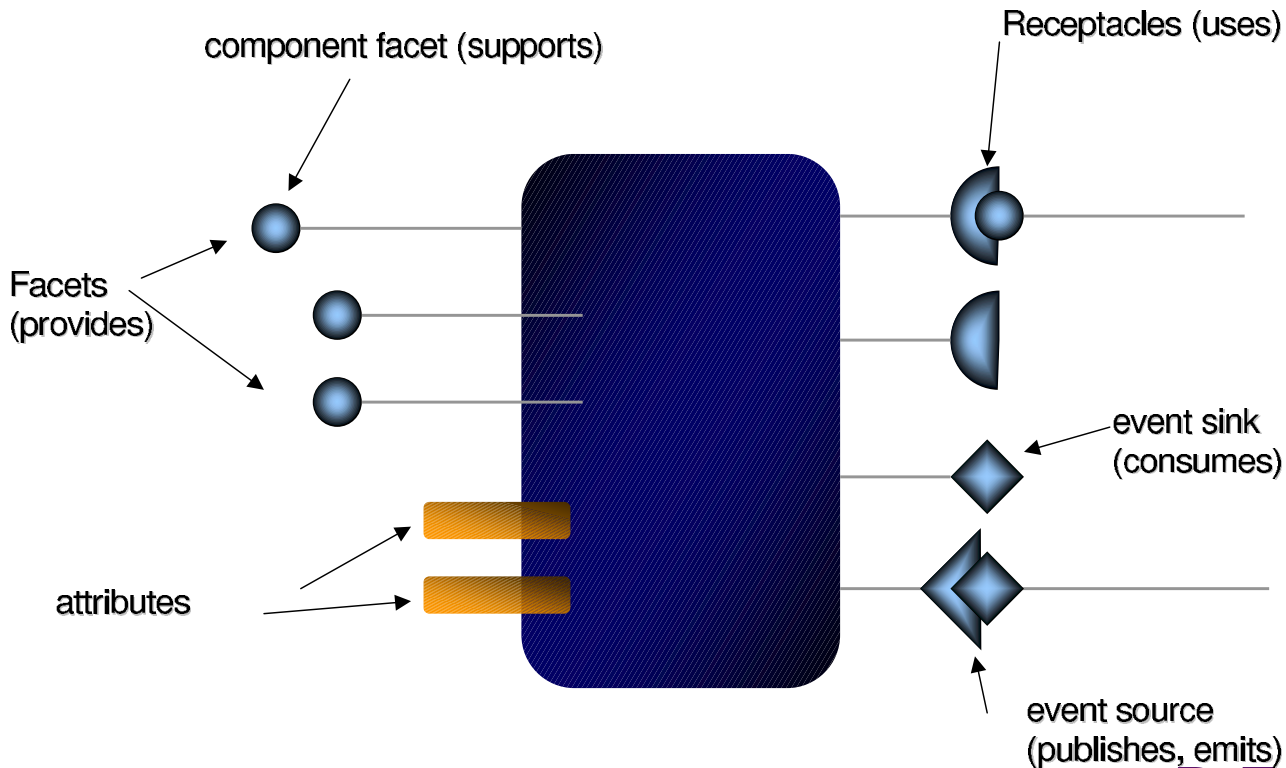
El modelo abstracto: visión general

- Permite la definición de interfaces y propiedades de componentes
- IDL3 extiende IDL2 con la descripción de
  - componentes con múltiples puertos (*ports*) de distinta índole
    - **facetas (*facets*)**: interfaces de invocación ofrecidas
      - primaria: la “interfaz equivalente” / la “faceta de component”; IDL3: ***supports***
      - secundarias: las otros facetas; IDL3: ***provides***
    - **receptáculos (*receptacles*)**: interfaces de invocación requeridas
      - de tipo sencillo; IDL3: ***uses***
      - de tipo múltiple; IDL3: ***uses multiple***
    - **fuentes (*sources*)**: interfaces de origen de eventos:
      - topología 1-n: como editor; IDL3: ***publishes***
      - topología 1-1: como emisor; IDL3: ***emits***
    - **sumideros (*sinks*)**: interfaces de destino de eventos:
      - IDL3: ***consumes***
      - no puede distinguir entre conexiones (*emits*) y suscripciones (*publishes*)
  - interconexiones entre estos puertos
  - atributos cuyos accesores/modificadores pueden lanzar excepciones

Modelos de componentes (III)

CCM Abstract Model





Flujos de eventos en CCM: editor / emisor

- Se distinguen dos tipos de interfaz de envío por canal de comunicación asíncrona:
  - editor (*publisher*):
    varios componentes pueden ser destino de los mensajes del canal
  - emisor (*emisor*):
    un solo componente puede ser el destino de los mensajes del canal

El modelo abstracto: homes

- **home**: meta-tipo nuevo de IDL3, e.g.

```
home aHome manages Server {...};
home CustomerHome manages Customer primaryKey SS# {...};
```

- Un *home* es un gestor de instancias de un tipo particular
  - patrón de diseño: *factory* (fábrica), e.g.

```
home aHome manages Server {
      factory mycreate(in long n)
};
```

  - patrón de diseño: *finder* (buscador), e.g.

```
home aHome manages Server {
      finder myop(in long m)
};
```

- Cada *home* sirve para crear/encontrar un tipo particular de componente
- Puede haber múltiples *homes* para el mismo tipo de componente

El modelo abstracto: API

- Components::Navigation
  - **`same_component`**: determina si una faceta pertenece a un componente
  - **`provide_facet`**: devuelve la referencia de objeto de la faceta
  - **`provide_all_facets`**: devuelve todas las referencias de objeto
  - **`provide_named_facets`**: devuelve las refs. de las facetas nombradas
  - **`describe_facets`**: devuelve una descripción de todas las facetas

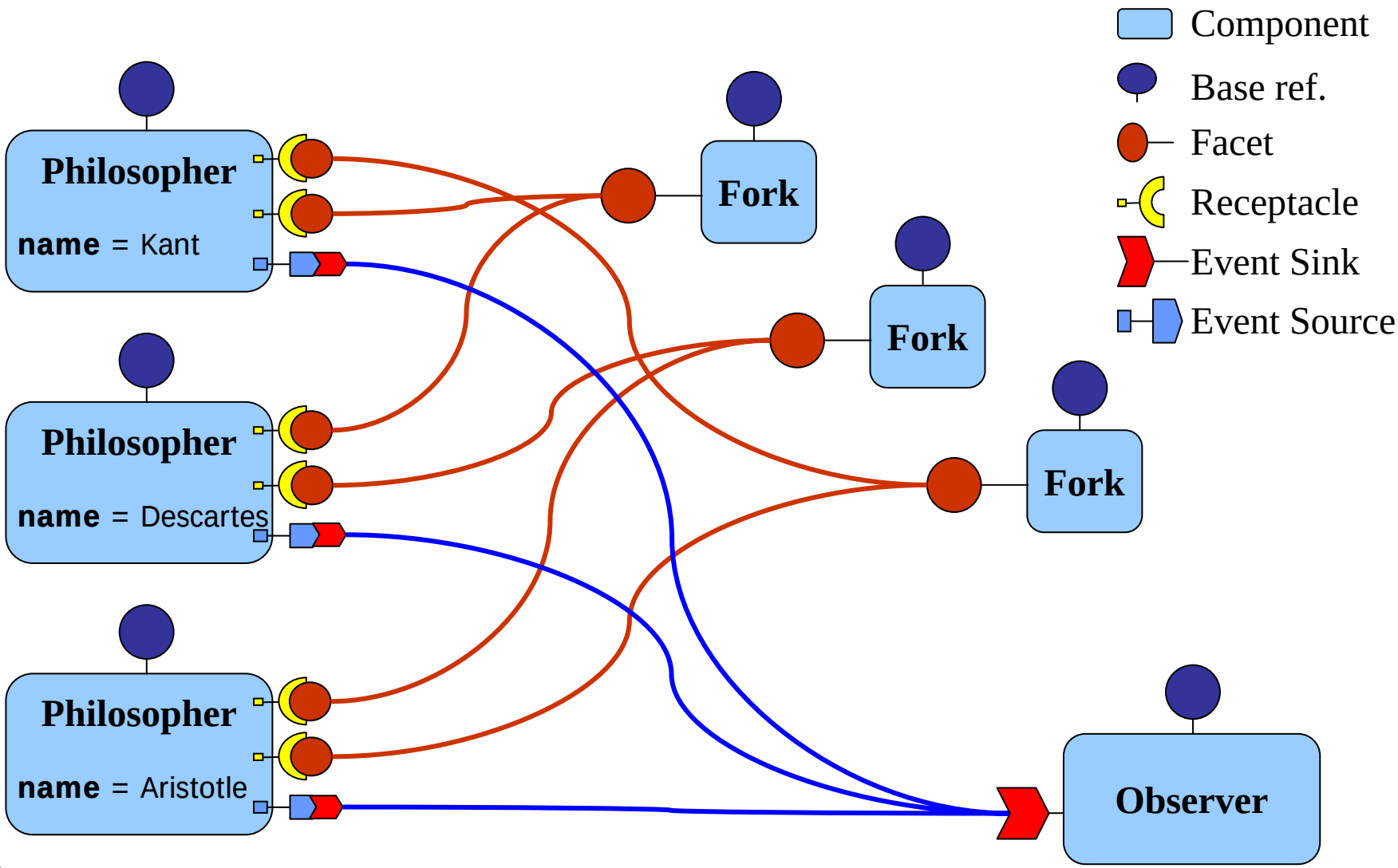
- Components::Receptacles
  - **`connect`**: conecta el receptáculo a la referencia de objeto proporcionada
  - **`disconnect`**: desconecta la referencia de objeto actualmente asociada desconecta según información del cookie, si es múltiple
  - **`get_connections`**: devuelve las descripciones de todas las connexiones

El modelo abstracto: API

- Components::Events
  - **`get_consumer`**: devuelve el sumidero (consume eventos) correspondiente
  - **`subscribe`**: suscribe una interfaz sumidero a un canal
  - **`unsubscribe`**: anula una suscripción
  - **`connect_consumer`**: conecta una interfaz sumidero a un canal
  - **`disconnect_consumer`**: desconecta una conexión
- Components::CCMObject
  - **`get_components_def`**: devuelve la ref. del objeto IR (∗) del componente
  - **`get_ccm_home`**: devuelve la ref. de la interfaz *home*
  - **`get_primary_key`**: devuelve la clave primaria si hay una asociada
  - **`configuration_complete`**: el componente se declara operativo
  - **`remove`**: destruye el componente
- CORBA::Object
  - **`get_component`**: dada una interfaz, devuelve el componente propietario (facilidad de introspección)

(∗) objeto IR = objeto del repositorio de interfaces

La comida de los filósofos en CCM

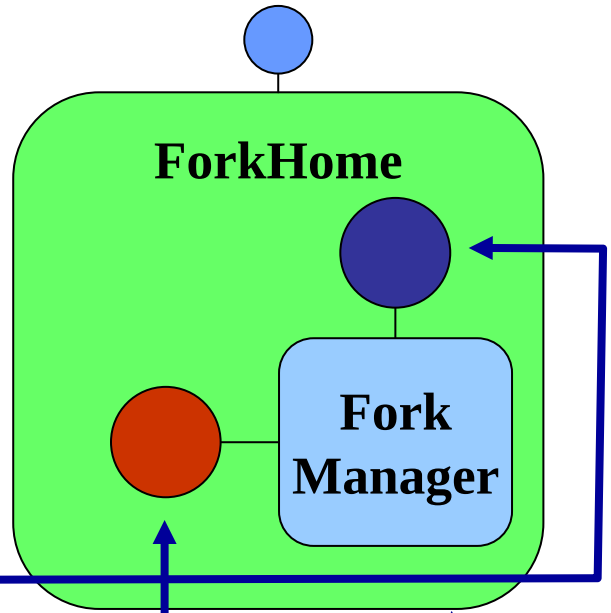


OMG IDL 3.0 para el ejemplo de los filosofos

```
// Importation of the Components module
// when access to OMG IDL definitions contained
// in the CCM Components module is required.
import Components;

module DiningPhilosophers
{
  // Sets the prefix of all these OMG IDL definitions.
  // Prefix generated Java mapping classes.
   typeprefix DiningPhilosophers "omg.org";
  . . .
};
```

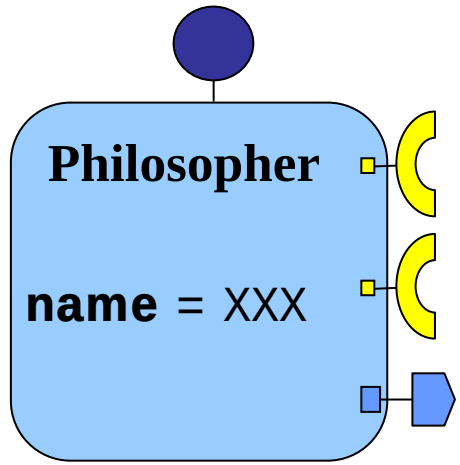
OMG IDL 3.0 para el ejemplo de los filosofos

```
exception InUse {};

interface Fork
{
   void get() raises (InUse);
   void release();
};

// The fork component.
component ForkManager
{
   // The fork facet used by philosophers
   provides Fork the_fork;
};

// Home for instantiating ForkManager components
home ForkHome manages ForkManager {};
```



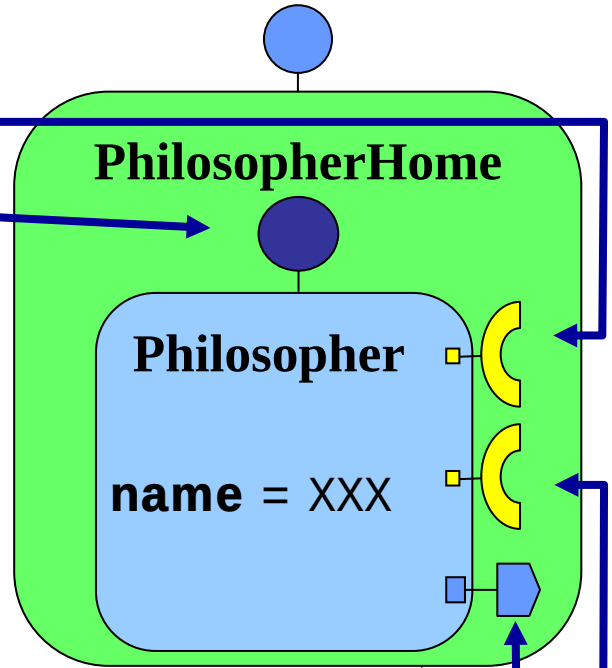
Los estados del filósofo

```
enum PhilosopherState
{
   EATING, THINKING, HUNGRY,
   STARVING, DEAD
};

eventtype StatusInfo
{
   public string name;
   public PhilosopherState state;
   public unsigned long ticks_since_last_meal;
   public boolean has_left_fork;
   public boolean has_right_fork;
};
```



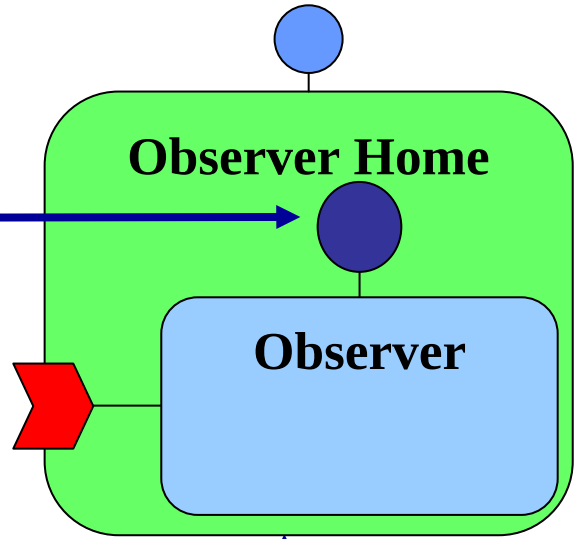
El componente filósofo

```
component Philosopher
{
   attribute string name;

   // The left fork receptacle.
   uses Fork left;

   // The right fork receptacle.
   uses Fork right;

   // The status info event source.
   publishes StatusInfo info;
};

home PhilosopherHome manages Philosopher {
    factory new(in string name);
};
```

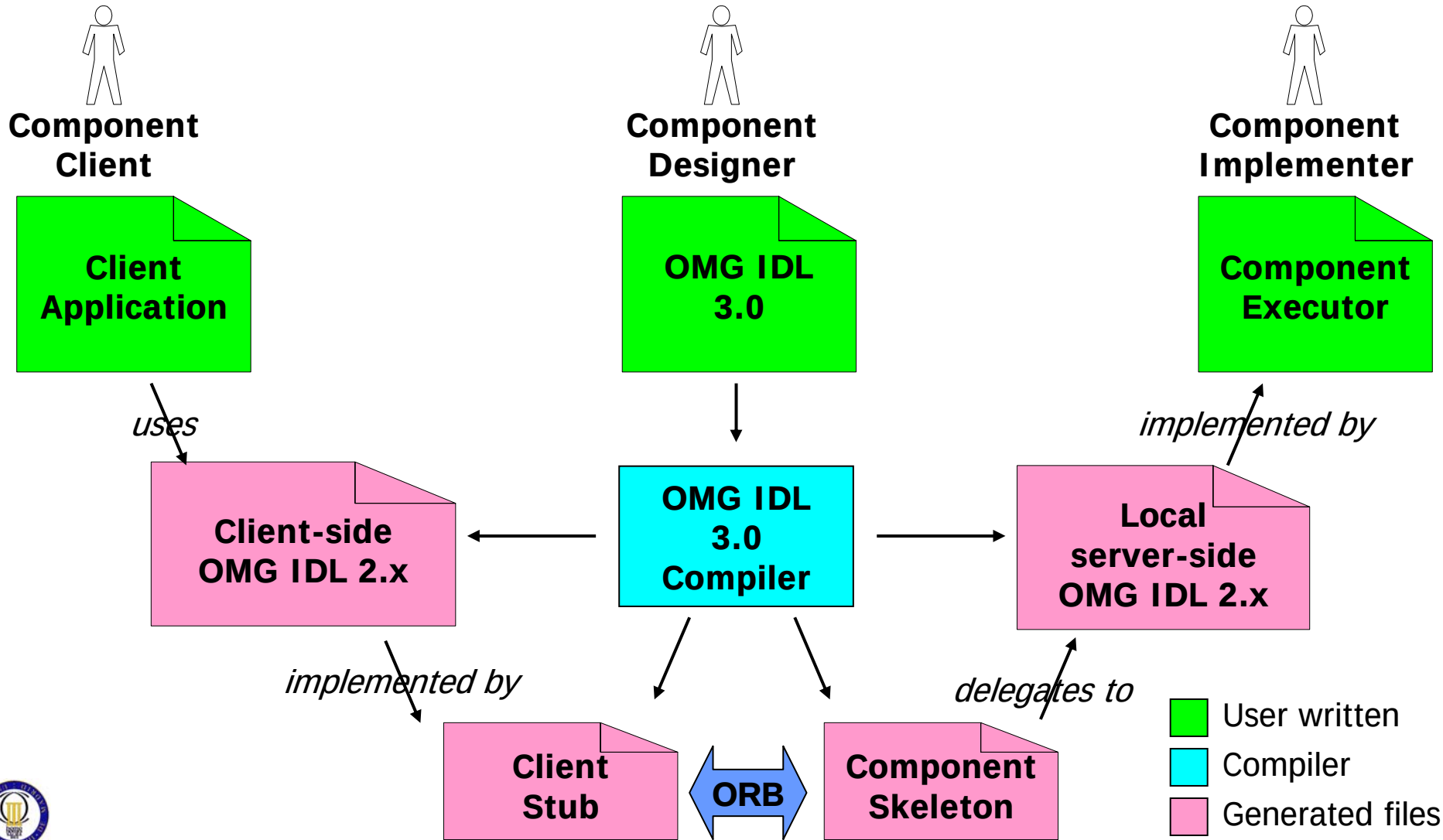


El componente Observer

```
component Observer
{
   // The status info sink port.
   consumes StatusInfo info;
};


// Home for instantiating observers.
home ObserverHome manages Observer {};
```



OMG IDL Mapping



Indice

- Introducción
- Modelo abstracto
- Modelo de programación
- Modelo de ejecución
- Modelo de despliegue
- Relación / comparación CCM - EJB

El modelo de programación: visión general

- Lenguaje de definición de implementaciones de componentes (CIDL)
  - para describir la estructura abstracta de un componente, tal como:
    - el conjunto de clases de implementación
    - su estado abstracto de persistencia (OMG PSDL)
- Armazón de implementación de componentes (CIF)
  - permite integrar
    - la parte funcional escrita por el diseñador
      incluye la implementación de las facetas y las fuentes
    - la parte no funcional generada a partir del IDL3 y del CIDL
- Compilación del IDL3 + CIDL produce:
  - un esqueleto/esbozo de la implementación
    - incluye parte no funcional
    - realizado encima de los APIs del contenedor
    - API de introspección: operaciones genéricas y especificas del componente
  - especificaciones en IDL2, que utilizarán los clientes del componente
  - un descriptor de despliegue en XML

Indice

- Introducción
- Modelo abstracto
- Modelo de programación
- Modelo de ejecución
- Modelo de despliegue
- Relación / comparación CCM - EJB

El modelo de ejecución: visión general

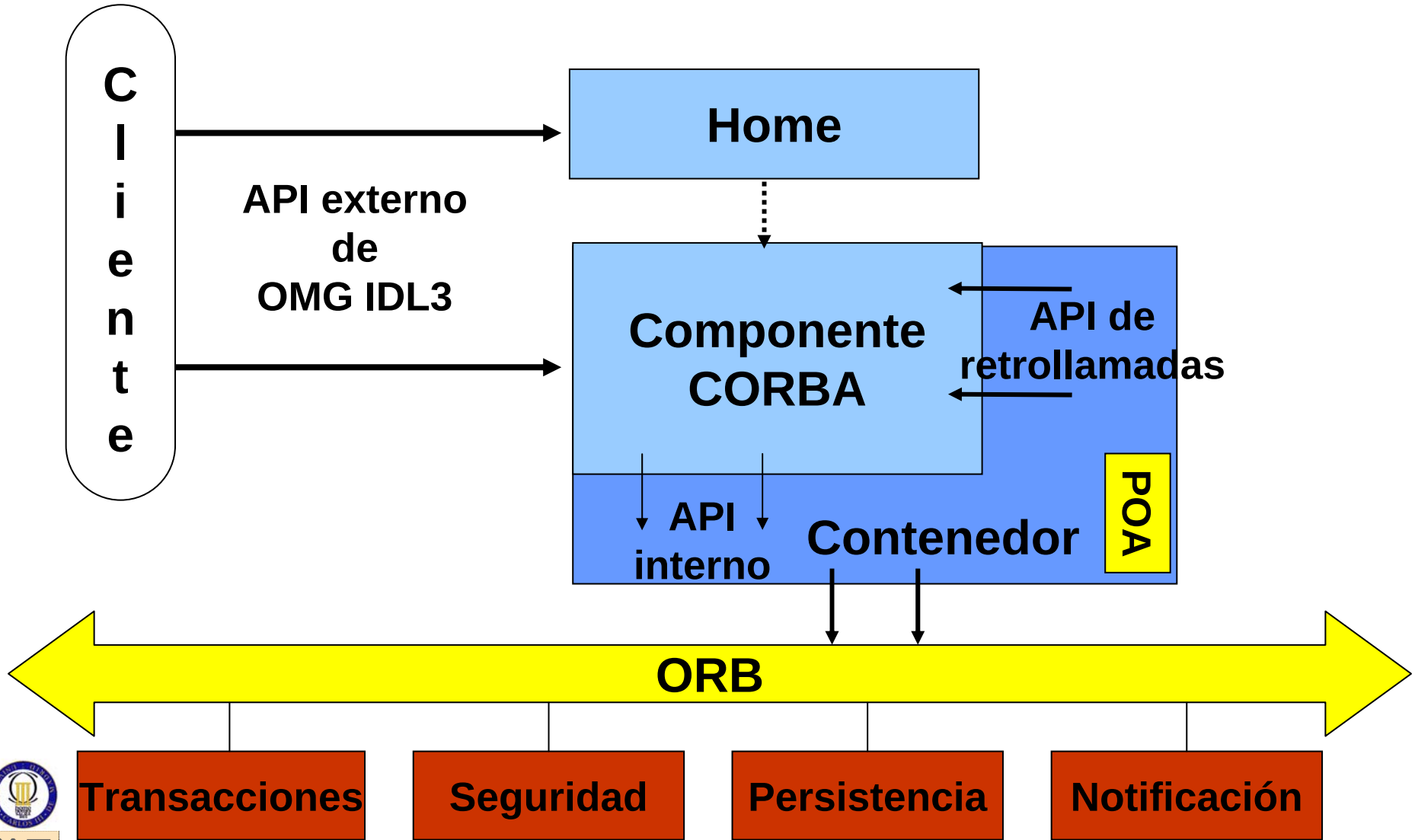
- Contenedor
  - entorno de ejecución para instancias de componentes y sus *homes*
  - puede verse como una especie de super-POA
    - encapsula uno o varios POA
  - oculta la complejidad de los servicios de sistema
    - seguridad
    - persistencia
    - transacciones
    - notificación etc.

    (gestiona los aspectos no funcionales)
- Servidor de aplicaciones
  - varios contenedores por servidor
  - varios componentes por contenedor
  - interfaz clara entre servidor de aplicaciones y contenedor
  - descarga dinámica de paquetes por el servidor

El modelo de ejecución: contenedor

- Para la interacción componente-contenedor, dos APIs:
  - API contenedor
  - API de retrollamadas invocadas por el contenedor sobre el cmpnte
- Contenedor permite
  - activación / desactivación automática
  - optimización del uso de recursos
  - soporte de persistencia
  - seguridad: control de acceso
  - gestión de transacciones
  - …
- Contenedor media el acceso al servicio de **`CosNotification`**
  - interfaces de eventos, fuentes y sumideros

La arquitectura del contenedor



Modelo de ejecución: categorías de componente

- Componente de servicio
  - con comportamiento no trivial
  - sin identidad
  - sin estado fuera de la transacción
  - modela la ejecución única e independiente de una “operación”
- Componente de sesión
  - con comportamiento no trivial
  - sin estado fuera de una interacción con un cliente
  - modela un estado transitorio que existe para una sesión

Modelo de ejecución: categorías de componente

- Componente de proceso
  - con comportamiento no trivial
  - con estado persistente (pero no visible para el cliente)
  - con identidad persistente (que puede ser visible para el cliente)
  - puede hacer *rollback* de sus cambios de estado
  - tiene que destruirse explícitamente
  - modela un proceso de duración larga (*workflow*)
- Componente de entidad
  - con comportamiento trivial (acceso y modificación de atributos)
  - con estado persistente
  - con identidad vía la clave primaria
  - puede hacer *rollback* de sus cambios de estado
  - tiene que destruirse explícitamente
  - modela entidades con una existencia no transitoria

Indice

- Introducción
- Modelo abstracto
- Modelo de programación
- Modelo de ejecución
- Modelo de despliegue
- Relación / comparación CCM - EJB

El modelo de despliegue: visión general

- Paquetes de software comprimido (como los jar de java)
- Descriptores escritos en OSD, un lenguaje XML:
  - descriptor de paquetes
    - información global: autor, licencia
    - información sobre interfaces, propiedades, dependencias, implementaciones
  - descriptor de componentes CORBA:
    - información sobre la implementación del componente
    - generado a partir del CIDL
  - descriptor de ensamblaje:
    - configuración estática inicial (qué componentes & cómo conectarlos)
  - descriptor de fichero de propiedades:
    - configuración dinámica (instancias)
- Despliegue y configuración automáticos donde ya están ejecutándose:
  - un objeto **`ComponentInstallation`**
  - un objeto **`AssemblyFactory`**

Indice

- Introducción
- Modelo abstracto
- Modelo de programación
- Modelo de ejecución
- Modelo de despliegue
- Relación / comparación CCM - EJB

CCM y EJB (I)

- CCM extiende el modelo de EJB a todos los niveles
  - modelo abstracto
    - modelo completo (interfaces de cuatro tipos, conexiones entre interfaces)
    - el lenguaje IDL3 (una extensión del IDL de CORBA 2)
  - modelo de programación
    - el lenguaje de definición de implementaciones CIDL
    - el armazón de implementación de componentes CIF
  - modelo de ejecución
    - interfaz bien definida entre contenedor y servidor de aplicaciones
  - modelo de despliegue
    - más genérico
    - pero descriptores de despliegue de CCM más complejos:
      - CCM: tienen que reflejar el lenguaje de programación
        EJB: presupone Java
      - CCM: tienen que reflejar el sistema operativo
        EJB: presupone máquina virtual Java

CCM y EJB (II)

- CCM es independiente del lenguaje de programación
- CCM es un superconjunto del modelo EJB
  - componentes de CORBA (CC) básicos:
    componentes que no utilizan los aspectos adicionales
  - componentes de CORBA (CC) extendidos
    componentes que utilizan los aspectos adicionales
- Para CCs básicos hechos en Java
  - uso obligado de los APIs de EJB 1.1
    - precedencia sobre *language mapping* IDL–Java del OMG
    - por tanto, un CC básico hecho en Java es un EJB
- CCM define un contenedor CORBA para EJBs
  - estandariza el uso de CORBA por contenedores EJB

CCM y EJB (III)

- Un CC extendido puede
  - ofrecer múltiples interfaces
    - CCM permite navegación entre interfaces
  - declarar interfaces requeridas
  - declarar interfaces de canales de eventos
  - utilizar *segmentos* múltiples
    - los segmentos permiten cortar en trozos un componente grande
    - distintos segmentos pueden tener propiedades de persistencia distintas