XI Semana de Ingeniería Eléctrica

Ingeniería de Software Basada en Componentes

Humberto Cervantes Maceda

6 Septiembre 2004

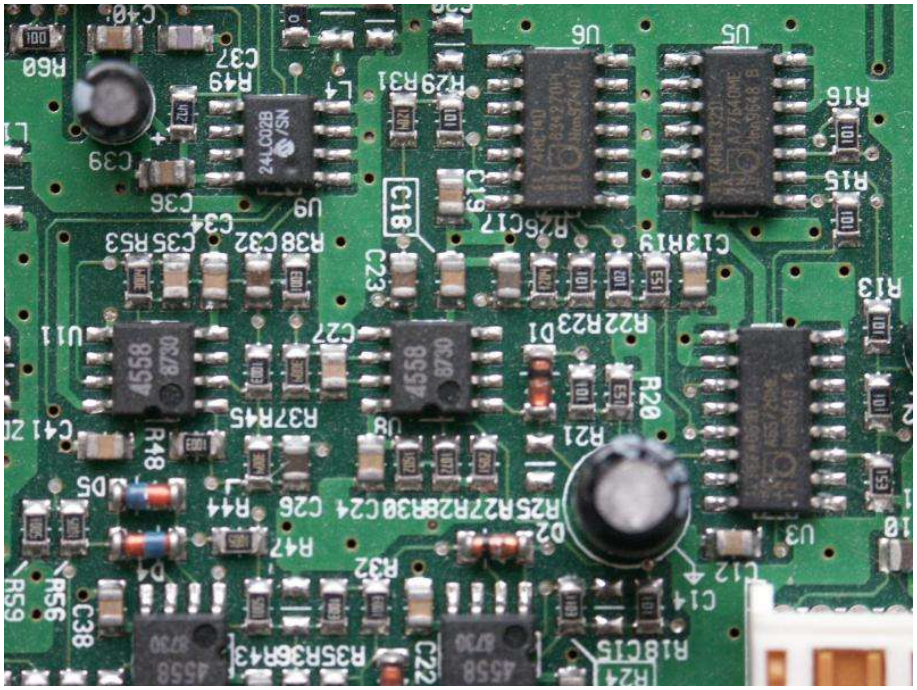
Índice

- Introducción
- Conceptos principales
- Tecnologías existentes
- Conclusiones

Índice

Motivaciones (1)

- **Construcción de un circuito electrónico (hardware)**



Circuito electrónico

Componentes

Motivaciones (2)

- **Características de los componentes electrónicos**
  - Son *reutilizables*
    - Un mismo componente puede ser utilizado para construir diversos circuitos

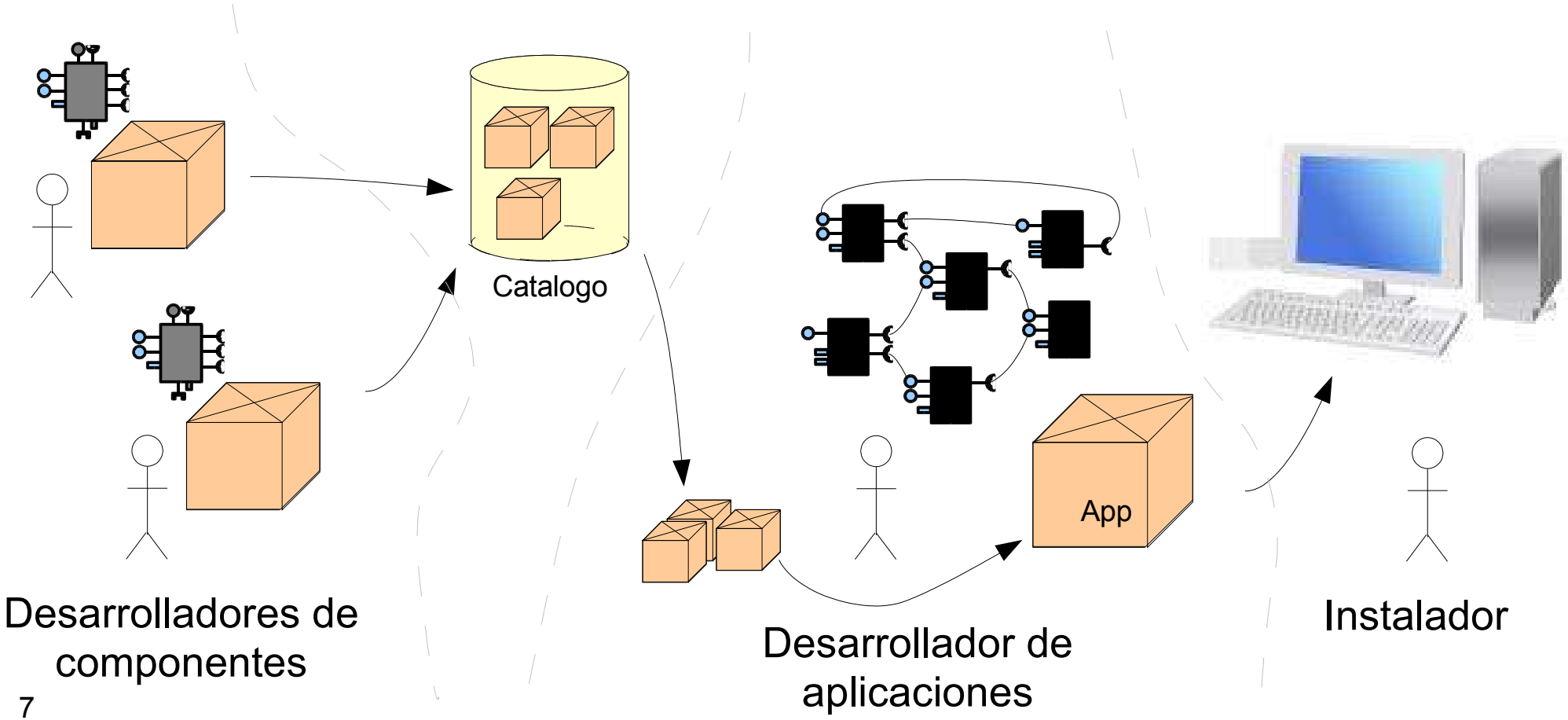
  - No es necesario conocer lo que tienen al interior para utilizarlos (“cajas negras”)
    - Acompañados de manuales que describen su funcionamiento
  - Están listados en catálogos
    - Por ejemplo: Familia 74xx de puertas lógicas

Motivaciones (3)

- **Que hay del lado del software?**
  - Muchas veces el software se construye 'desde cero' pues hay poca reutilización
- **Necesidades de la industria de construcción de software**
  - Reducción del tiempo y costos de desarrollo
  - Especialización de los actores participantes
  - Creación de aplicaciones más complejas
  - Soporte de la evolución

Motivaciones (4)

- **Objetivo de la I.S. a base de componentes**
  - Soportar la construcción de software de forma similar a como se construye el hardware



Índice

- **Introducción**
  - Motivaciones
  - Antecedentes
  - Síntesis
- **Conceptos principales**
- **Tecnologías existentes**
- **Conclusiones**

Antecedentes

- **Que significa reutilización?**
  - Representa la utilización de artefactos de software en la resolución de múltiples problemas.
  - Una misma entidad de software es usada para construir distintas aplicaciones, evitando la construcción 'desde cero'.
- **Antecedentes de los componentes**
  - Librerías
  - Módulos
  - Programación orientada a objetos
  - Frameworks y plataformas extensibles por plug-ins

Librerías

- **Características**
  - Colección de funciones o procedimientos compilados de forma independiente
  - No son programas completos, mas bien conjuntos de código de soporte reutilizables
  - Incorporación a un programa a través del *ligado,* estático o dinámico
- **Ejemplos del lenguaje C (1971):**
  - math.h funciones matemáticas
  - stdio.h entrada y salida estándar
- **Forma más simple de reutilización**

Módulos (1)

- **Características**
  - Unidades de compilación que contienen funciones y estructuras de datos típicamente asociados a una tarea particular
  - Basados en la separación entre interfase e implementación
- **Limitaciones**
  - Poco configurables (requieren recompilación)
  - Dependencias difíciles de modificar

Módulos (2)

- **Un ejemplo en Modula-2 (1979)**

```
DEFINITION MODULE UnModulo;
  TYPE
     ptTipo; (* un tipo de apuntador *)
  PROCEDURE Funcion( VAR v : ptTipo ) : INTEGER;
END UnModulo.
```

Interfase
- fija

```
IMPLEMENTATION MODULE UnModulo;
  IMPORT OtroModulo
  CONST
      MAX = 10; (* privado *)
  TYPE
     ptTipo = POINTER TO Tipo;
  PROCEDURE Funcion( VAR v : ptTipo );
  BEGIN
    (* Cuerpo de la función *)
    ...
  END Funcion;

BEGIN (* UnModulo *)
  (* vacio *)
END UnModulo.
```

Implementación
- puede cambiar
- incluye dependencias hacia otros módulos

Programación Orientada a Objetos (1)

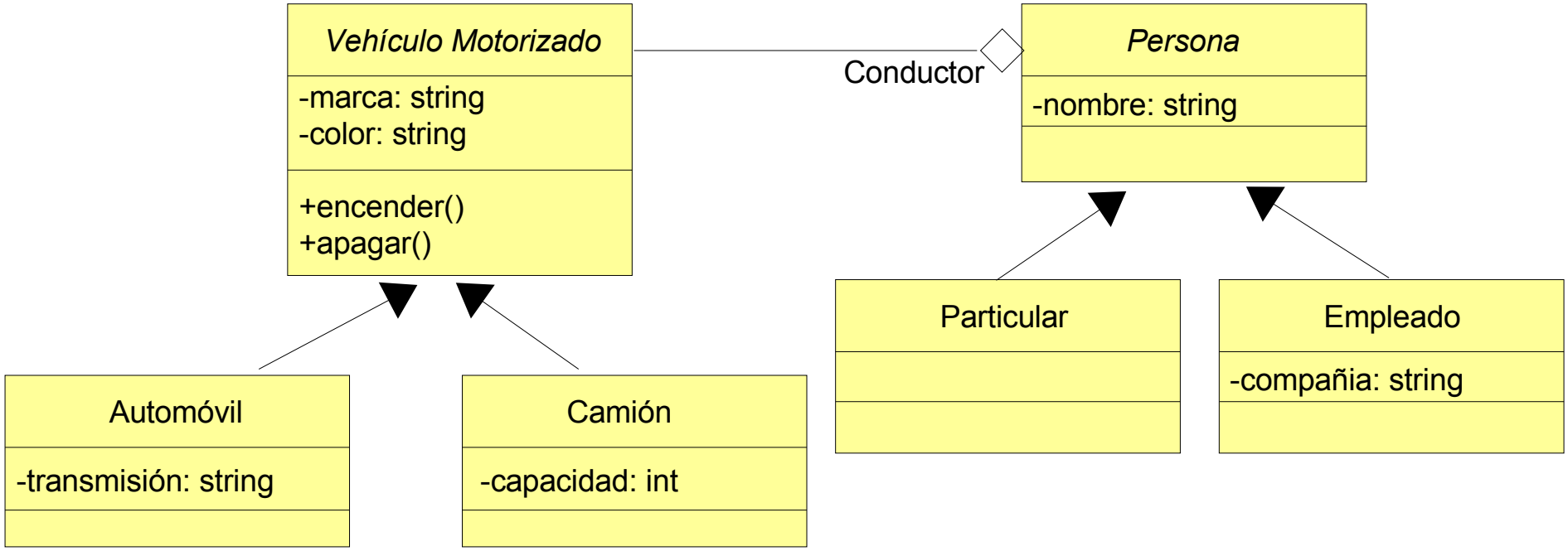
- **Objeto**
  - Entidad que encapsula datos (estado) y comportamiento (operaciones sobre estado) y existe en tiempo de ejecución
  - Creado a partir de la abstracción de un concepto que existe en la realidad
    - Ejemplo: Vehículo
  - Varias copias de un objeto - llamadas *instancias* - pueden existir
    - Creadas a partir de una *clase*
    - Identificadas de manera única

Programación Orientada a Objetos (2)

- **Herencia**
  - Una clase hereda y especializa estado y comportamiento de otra
    - Ejemplo: Automóvil
  - Mecanismo de reutilización
- **Polimorfismo**
  - Asociación entre instancias de clases pueden variar según el contexto de ejecución

Programación Orientada a Objetos (3)

- **Ejemplo**



Automóvil conducido por particular

Automóvil conducido por empleado

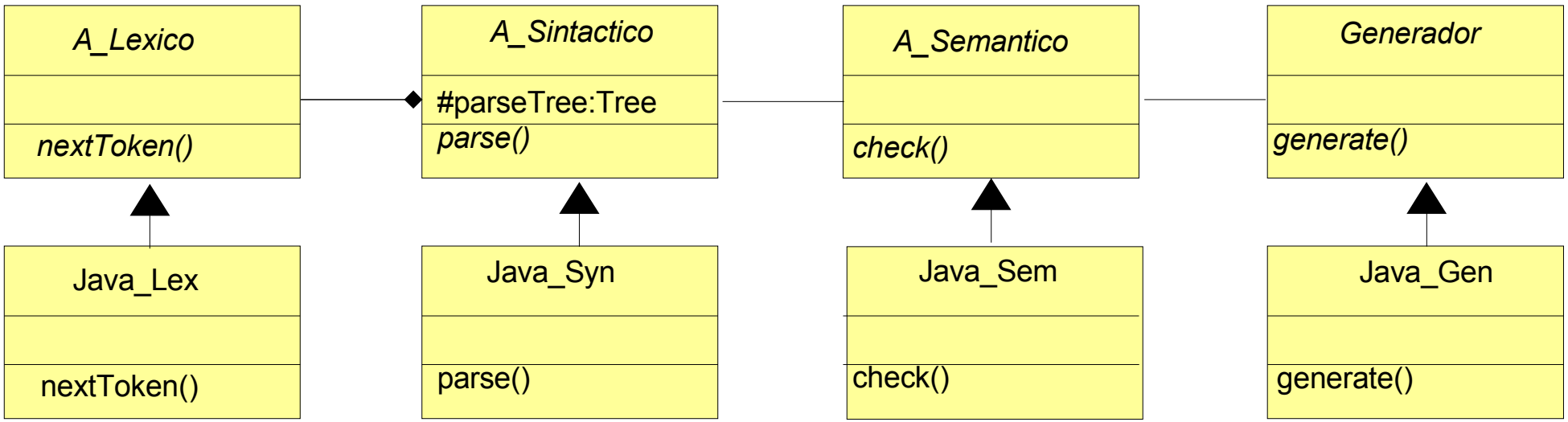
Camión conducido por empleado

Programación Orientada a Objetos (4)

- **Limitaciones**
  - Herencia dificulta evolución de clases de base sin impactar subclases
  - Dependencias ocultas dentro del código
  - Poco soporte para empaquetar y distribuir objetos de manera binaria

Frameworks (1)

- **Características**
  - Conjunto de clases que definen un diseño abstracto que permite resolver problemas ligados a un dominio particular
  - Reutilización a través de la herencia a nivel superior que la clase

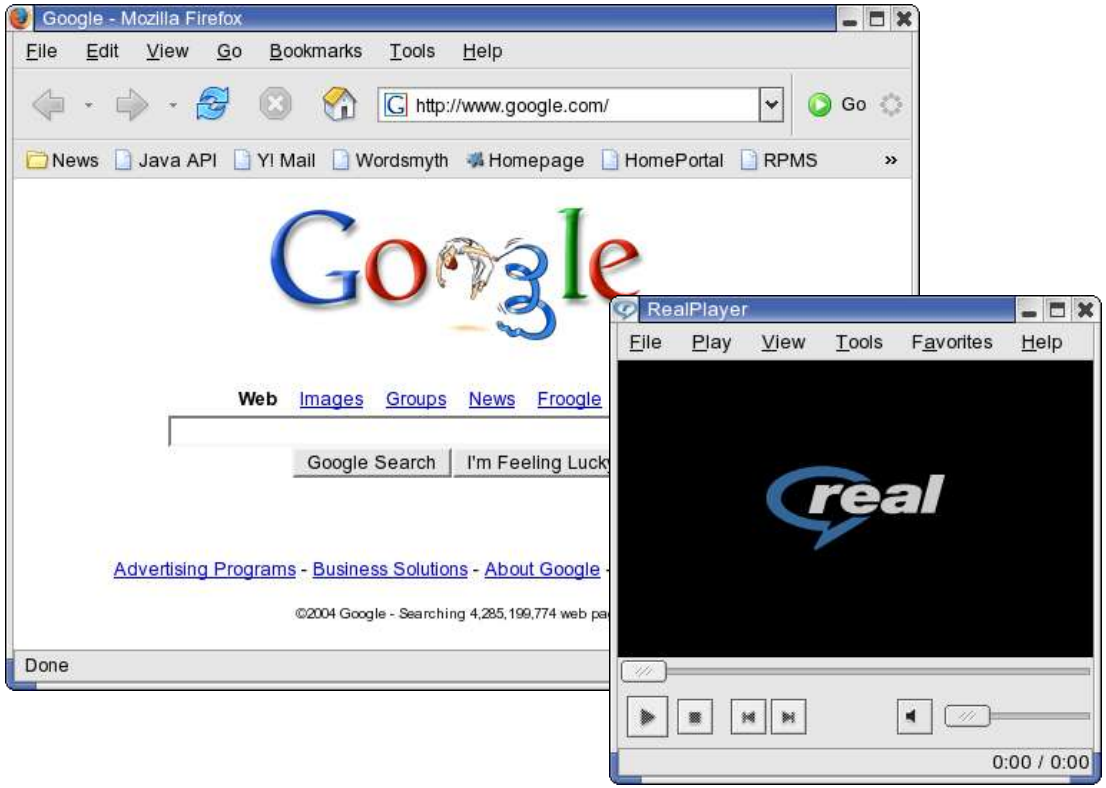


Frameworks (2)

- **Limitaciones**
  - Requiere creación de sub-clases
  - Difícil de reutilizar pues se necesita un buen conocimiento del framework antes de poder utilizarlo
  - Dificultad de evolución pues cambios en clases de base repercuten en sub-clases

Plataformas extensibles (1)

- **Características**
  - Sistemas que soportan la introducción tardía de extensiones
    - Aplicaciones extensibles por *plug-ins* : navegadores, IDEs, Winamp, etc...



NP_Plugin <<interface>>
+NP_Initialize() +NPP_New() +NPP_Destroy() +NPP_Shutdown()

Plataformas extensibles (2)

- **Características**
  - Basadas en existencia de una interfase común y del cargado dinámico
  - Plug-ins transferibles y desplegables de forma independiente
- **Limitaciones**
  - Extensiones solo en puntos bien definidos
  - Plug-ins poco reutilizables, aunque en teoría se podrían usar en distintas plataformas

Índice

- **Introducción**
  - Motivaciones
  - Antecedentes
  - Síntesis
- **Conceptos principales**
- **Tecnologías existentes**
- **Conclusiones**

Síntesis

- **Idea de reutilización presente desde hace bastante tiempo**
  - Sin embargo, las técnicas existentes solo han permitido realizarlo a una escala limitada
- **Otra limitación ha sido la falta de un medio para crear un mercado de componentes**
  - Catalogo de componentes
  - Medio que permita transferirlos

Índice

- **Introducción**
- **Conceptos principales**
  - Definición
    - Características de los componentes
- **Tecnologías existentes**
- **Conclusiones**

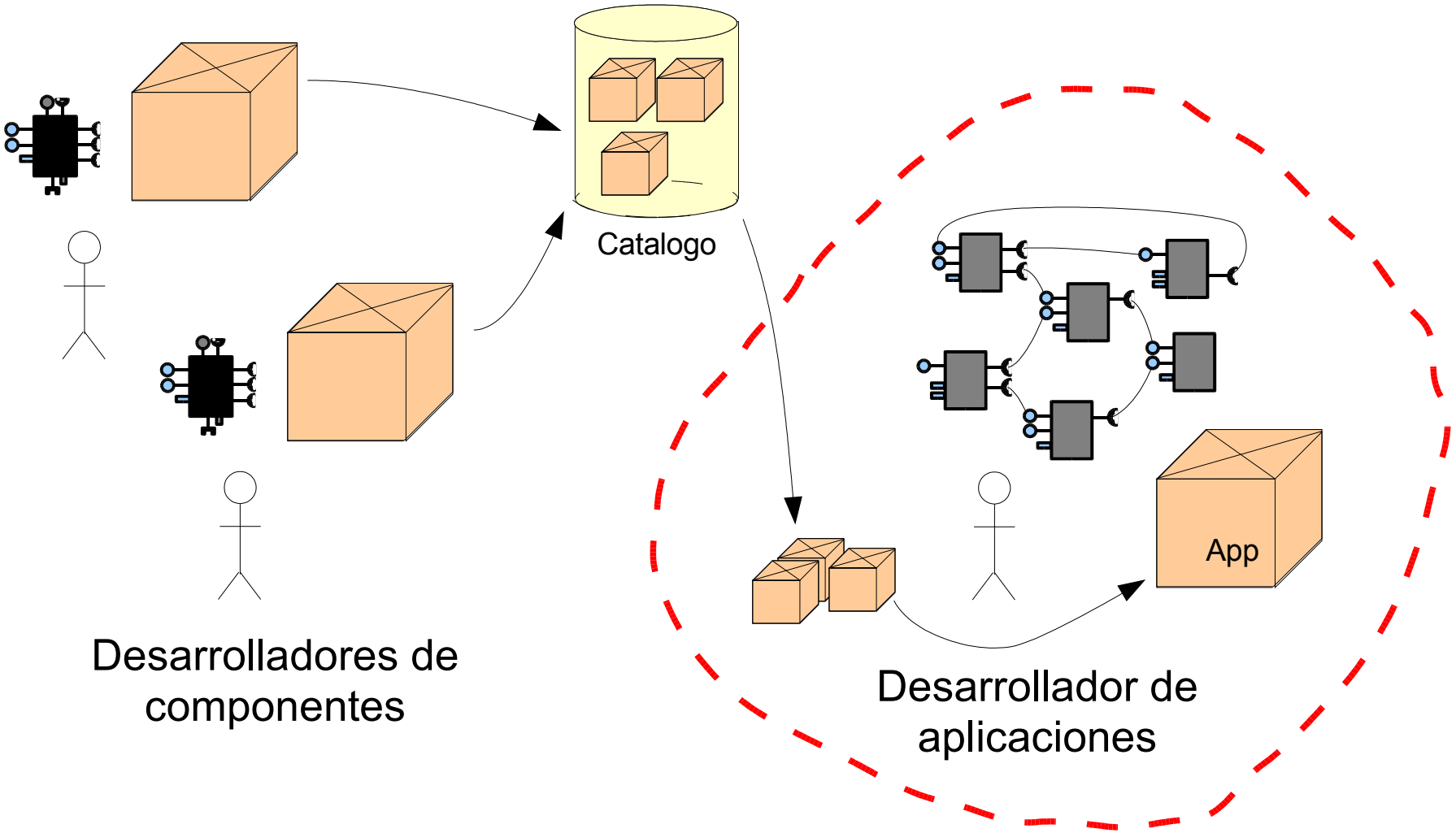
Definición

- **No existe una definición consensual de lo que es un componente!**

- **Definición de Clemens Szyperski:**
  - "Un componente de software es una **unidad binaria** de composición con **interfaces contractualmente definidas y dependencias explicitas** con respecto a un contexto. Un componente de software puede ser **desplegado de manera independiente** y es sujeto a **composición por terceros.**"

Composición por terceros

- La composición por terceros es el punto clave de la definición



Índice

- **Introducción**
- **Conceptos principales**
  - Definición
  - Características de los componentes
- **Tecnologías existentes**
- **Conclusiones**

Diagrama de un componente

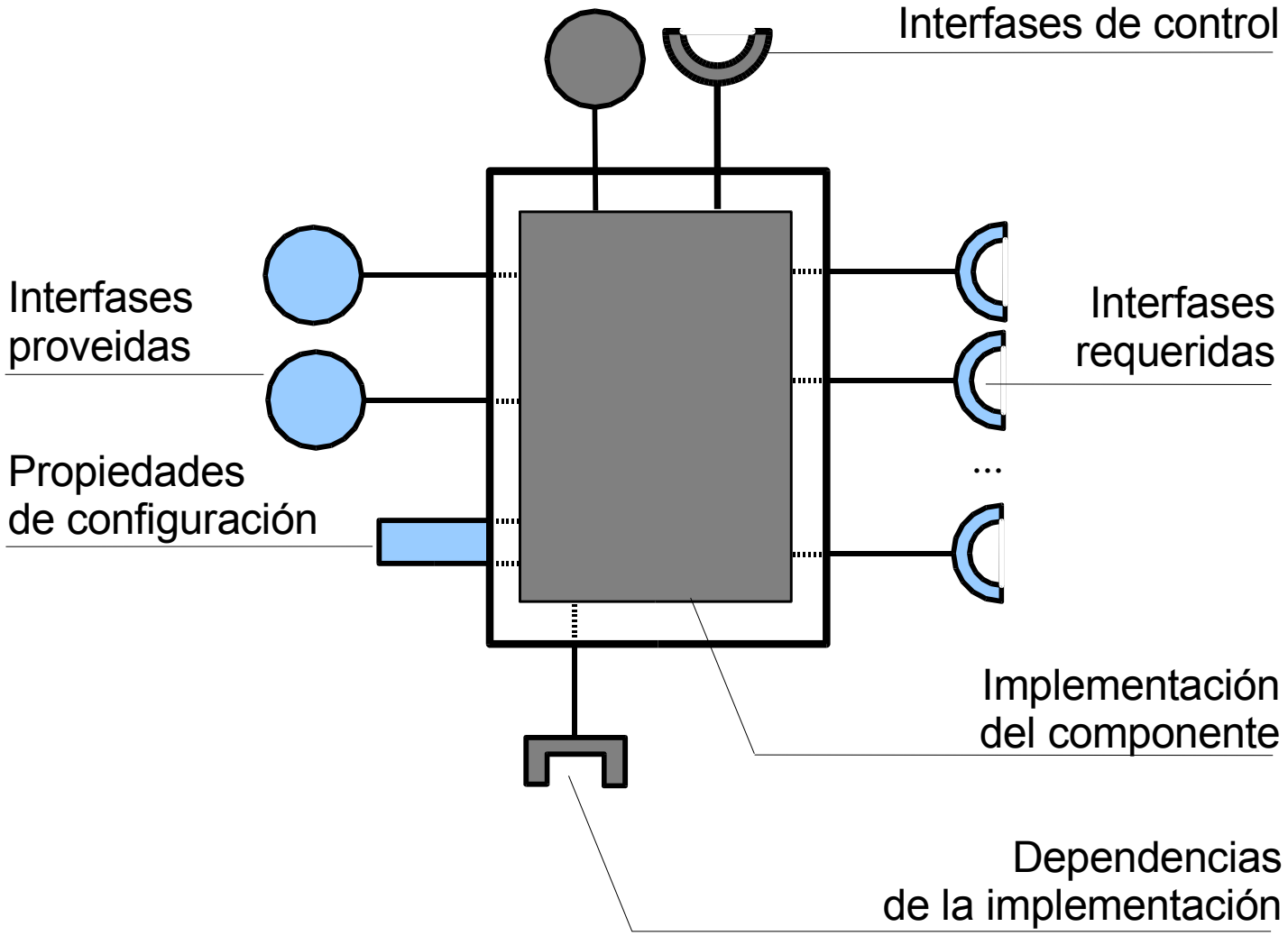
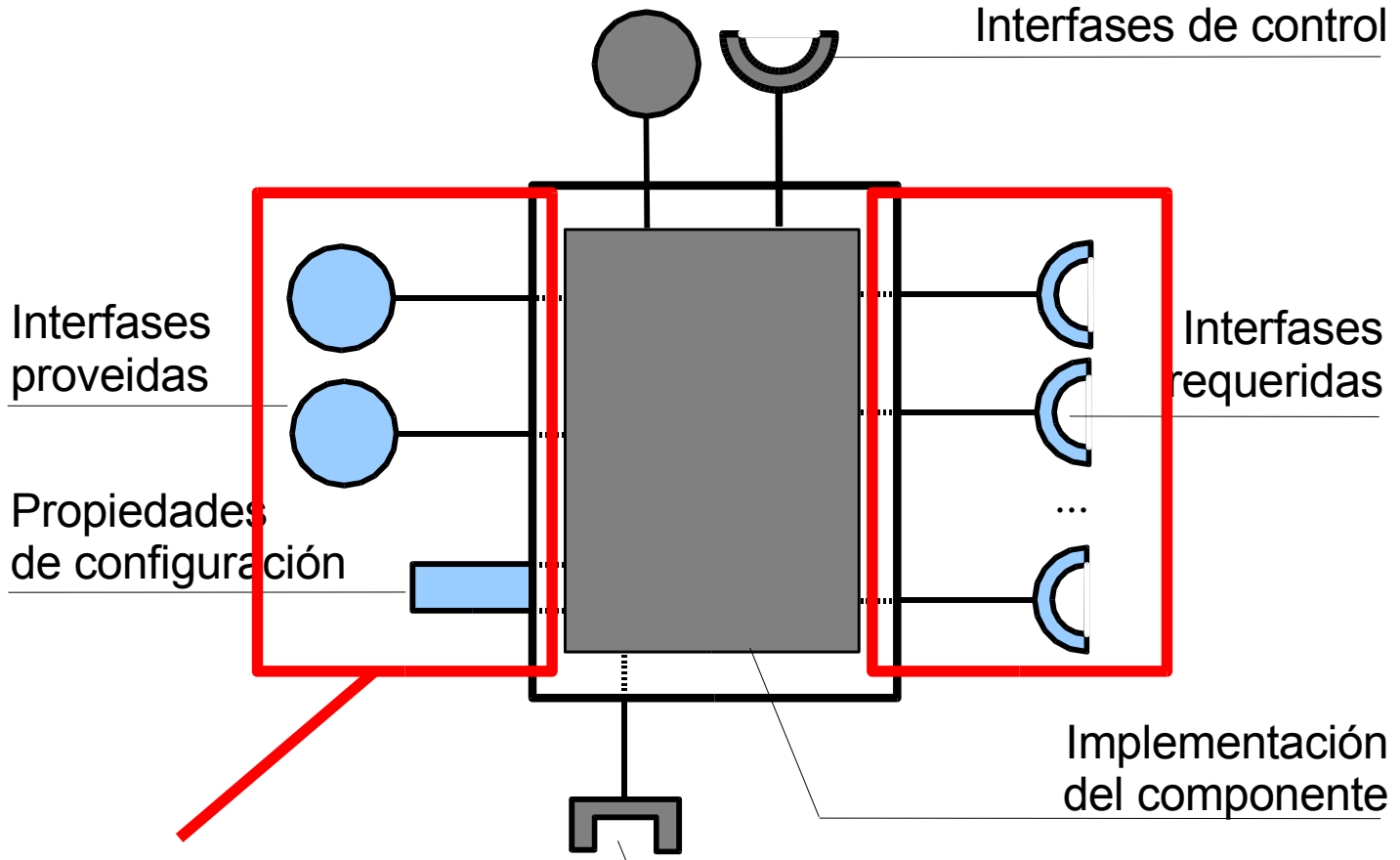


Diagrama de un componente



Permiten la composición

Composición

- **Típicamente la composición se realiza conectando instancias de componentes**
  - Lenguaje especializado o estándar
  - Visualmente

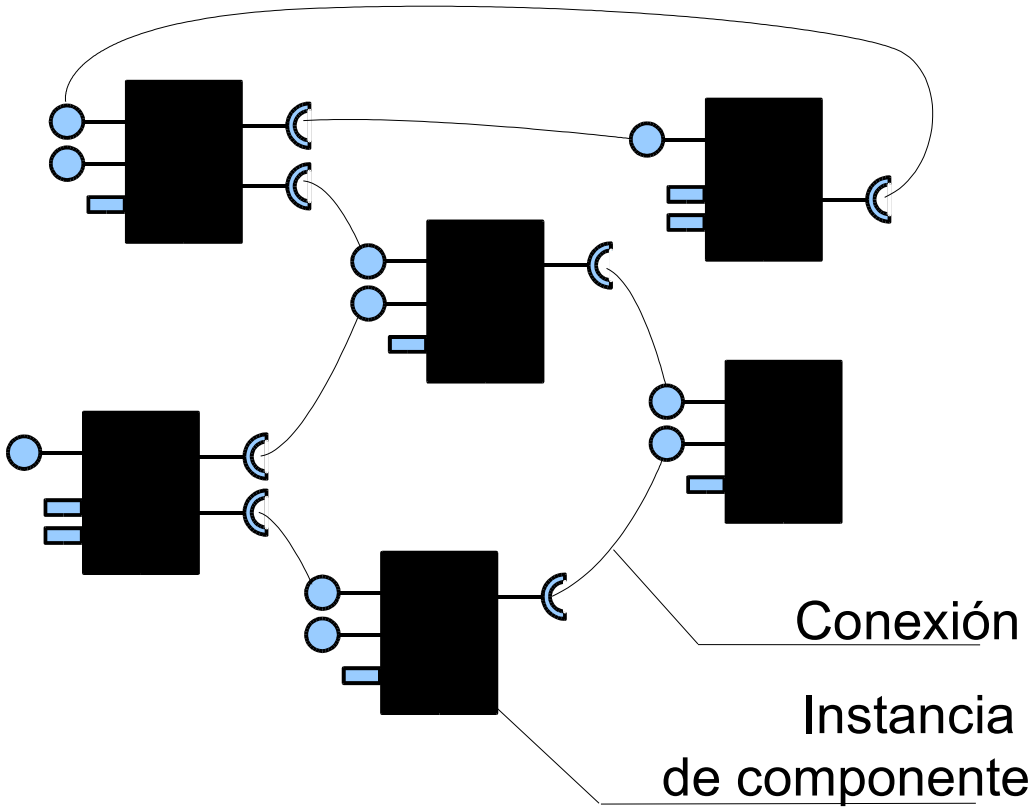
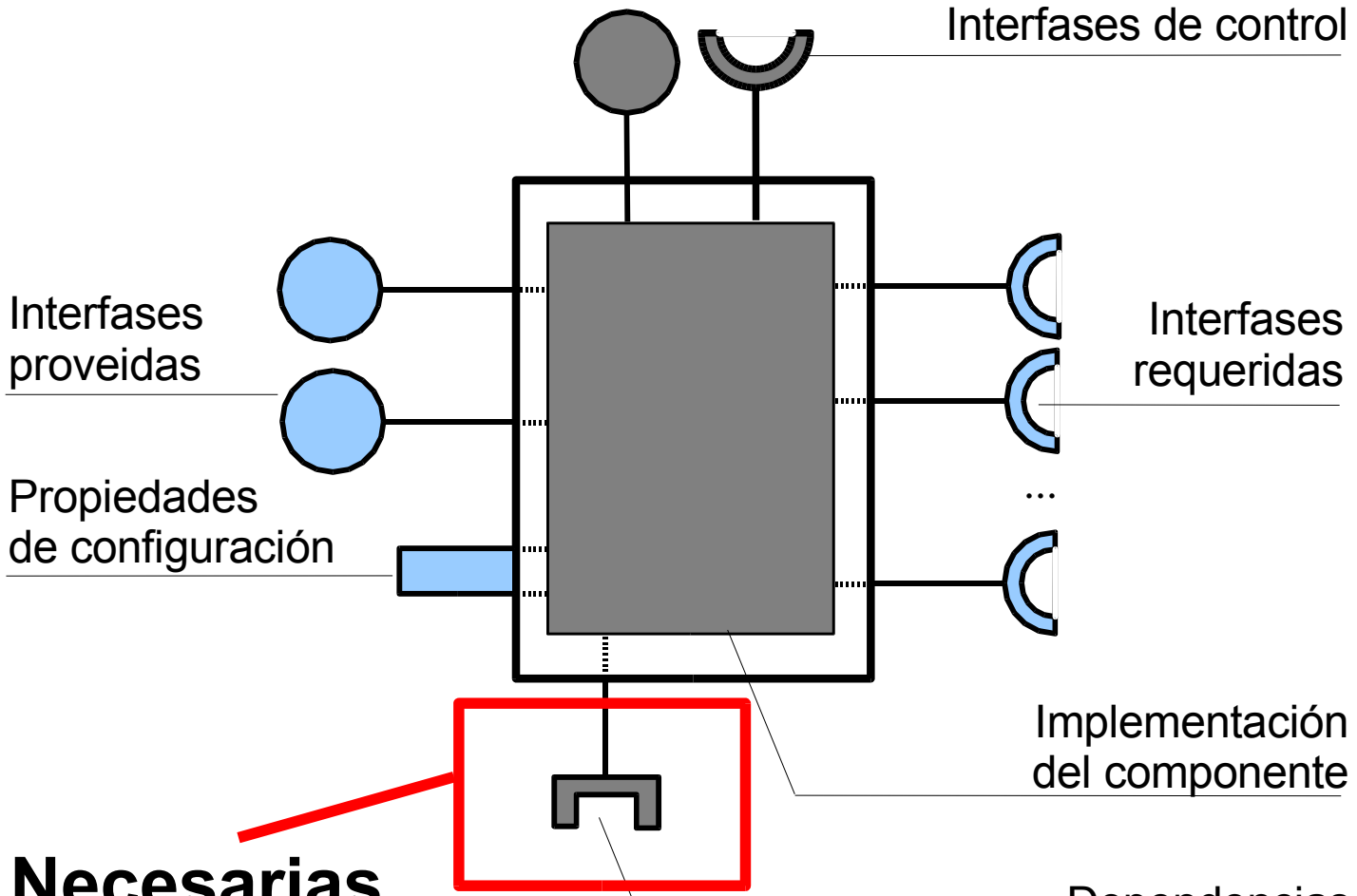
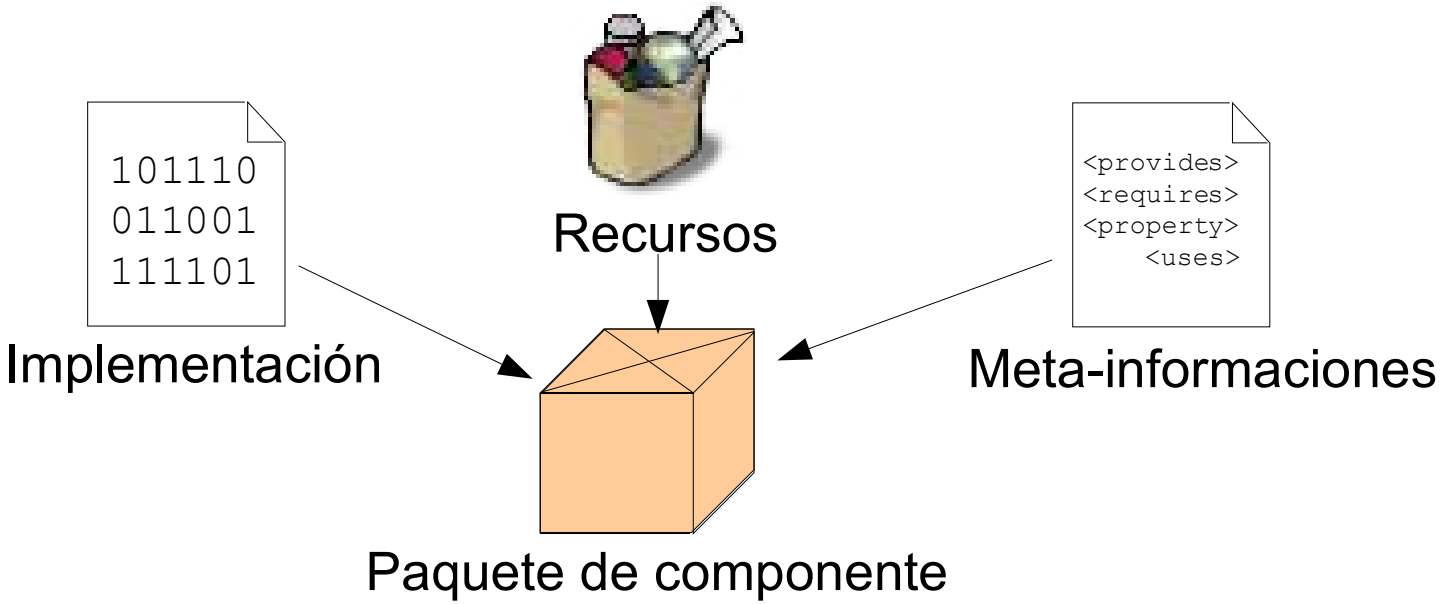


Diagrama de un componente



Paquete de componente

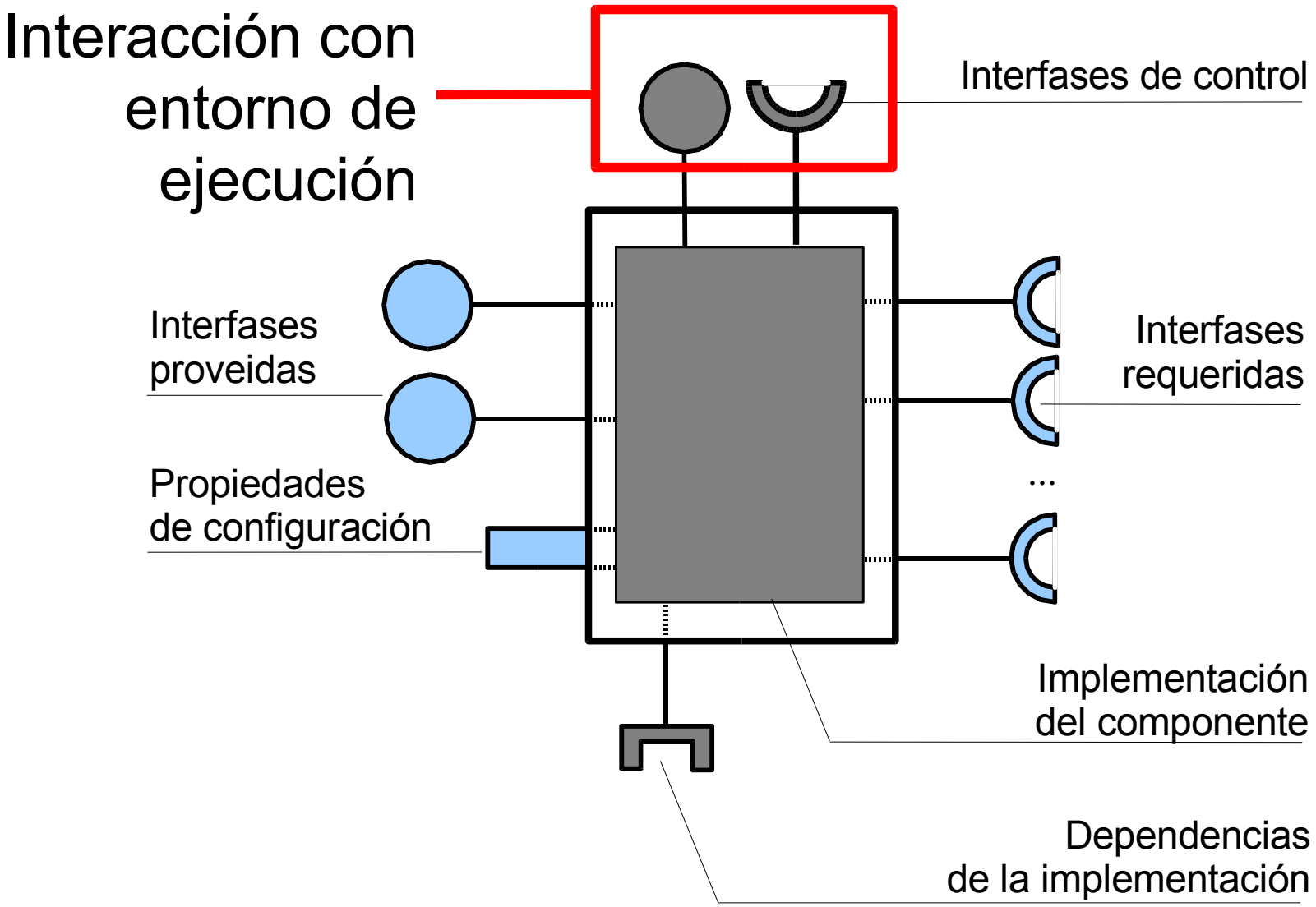


- **Despliegue independiente**
  - Un componente es contenido en un paquete que contiene todo lo necesario a excepción de lo que se define explícitamente como una dependencia.
- **Transferencia de forma binaria**
  - Protección del código fuente

Entorno de ejecución

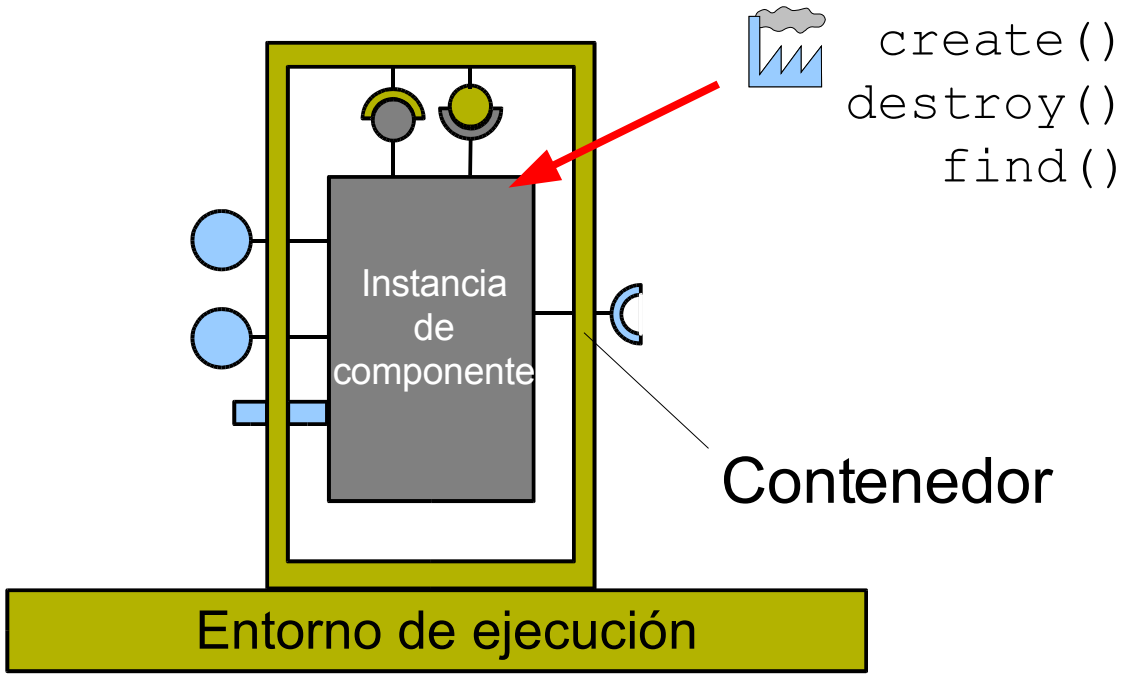
- **Asociado a un modelo de componentes**
  - Un modelo de componentes define las características de los componentes y la manera como interactúan
- **Provee soporte en tiempo de ejecución**
  - Creación de instancias
  - Control del ciclo de vida
  - Acceso a la infraestructura
  - Aspectos no funcionales

Diagrama de un componente



Instancias

- **En tiempo de ejecución**
  - Instancias creadas a partir de fabricas
  - Manejadas por un contenedor
    - Control del ciclo de vida
    - Aspectos no funcionales (distribución, seguridad,...)



Síntesis

- **El aspecto mas importante es la composición por terceros**
  - Componente describe de forma explicita lo que provee y lo que requiere
- **Tres niveles distintos**
  - Componente
  - Instancias de componente
  - Paquete de componente
- **Un entorno de ejecución esta asociado a cada modelo de componentes**

Índice

- **Introducción**
- **Conceptos principales**
- **Tecnologías existentes**
  - COM (1995)
  - JavaBeans (1997)
  - EJB (1998)
  - CCM (1999)
  - OSGi (2000)
- **Conclusiones**

COM: Component Object Model

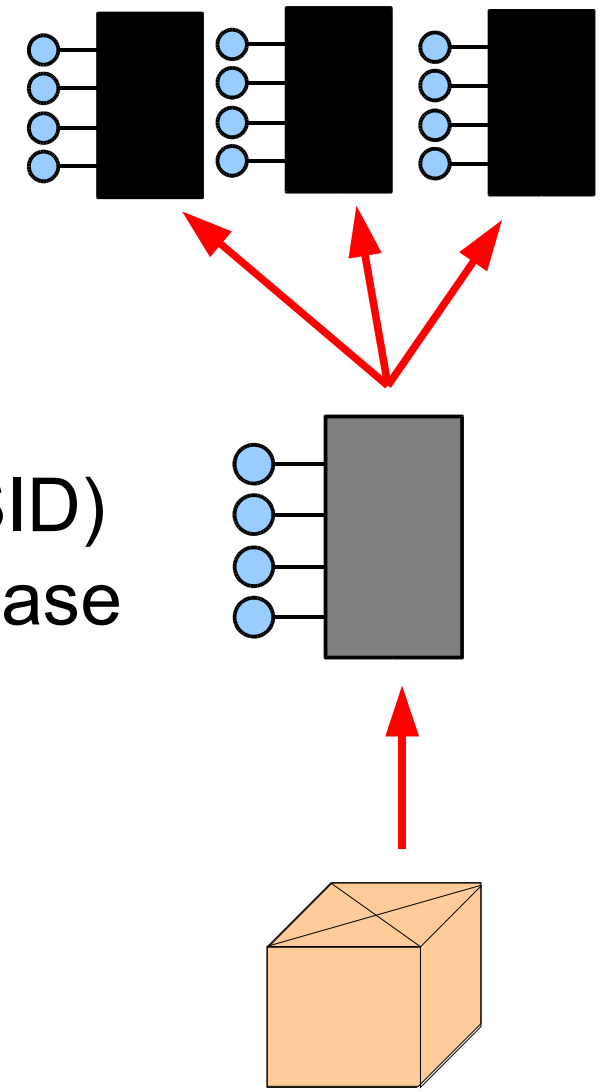
- **Microsoft ≈ 1995**
  - "COM es una arquitectura a base de componentes que permite construir aplicaciones y sistemas a partir de componentes proveidos por distintos vendedores"
  - Interoperabilidad binaria
    - Componentes programados en diferentes lenguajes pueden interactuar

Ejemplo de aplicación COM

- El “MediaPlayer” utiliza un componente navegador para desplegar paginas web.
- Navegador proveido por IE o por Netscape

Entidades COM

- **Instancias: Objetos COM**
  - Implementan múltiples interfaces
  - Soportan introspección
- **Componentes: Clases COM**
  - Identificadas de manera única (CLSID)
  - Una fabrica esta asociada a cada clase
  - Describen interfases proveidas
- **Paquetes: DLLs o EXEs**
- **Entorno de ejecución**
  - Localización, creación de instancias

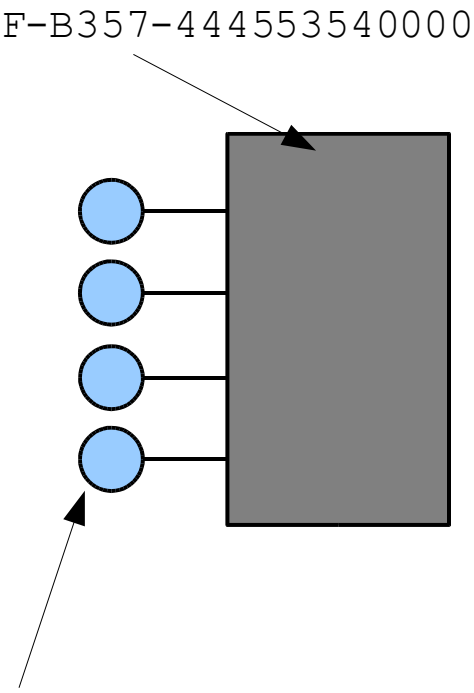


Clases e interfases

- **Clases COM**
  - Publicadas en el Registry de Windows

```
HKEY_CLASSES_ROOT\CLSID\{DDF7D820-8355-11CF-B357-444553540000}
                       \InprocServer32=c:\Progs\IE\explorer.dll
```

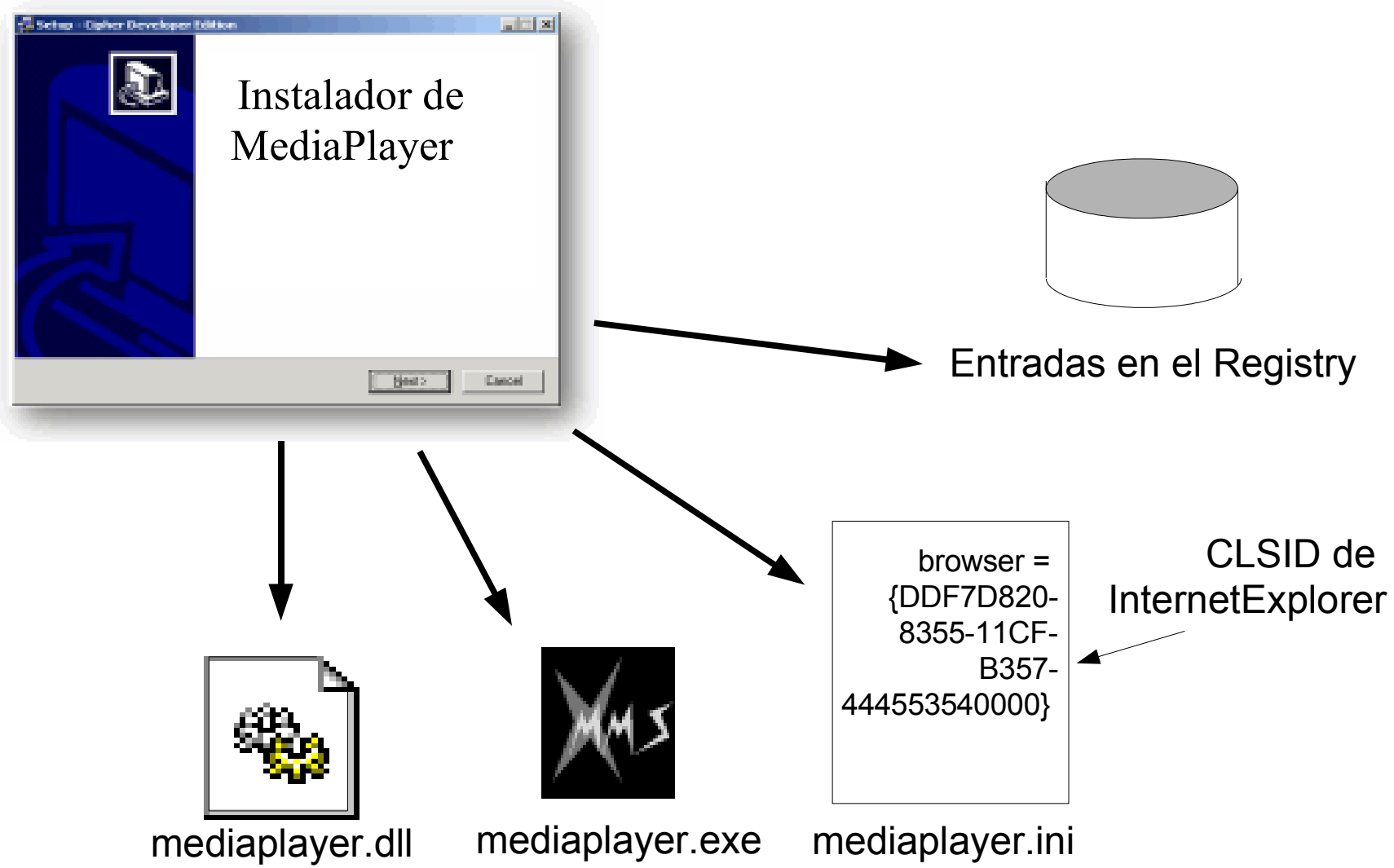
- **Interfases COM**
  - Identificadas e Inmutables
  - Descritas en IDL
  - Heredan de `IUnknown`
    - Provee metodo `QueryInterface()`




```
interface iBrowser extends iUnknown
{
    GUID={12345678-0000-0000-0000-123456789ABC}
    GraphicObject display(String URL);
}
```

COM

- **Instalación de la aplicación**



Durante la ejecución

Acción “Lanzar Navegador”

Durante la ejecución

Acción “Lanzar Navegador”

browser = {DDF7D820-8355-11CF-B357-444553540000}

Lectura del CLSID

mediaplayer.ini

Durante la ejecución

CoCreateInstance

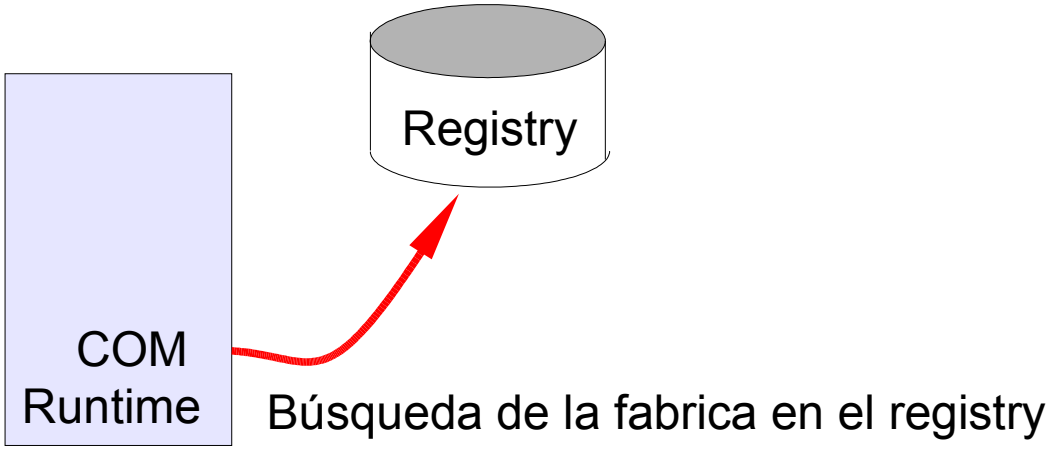
COM Runtime

Registry

browser = {DDF7D820-8355-11CF-B357-444553540000}

mediaplayer.ini

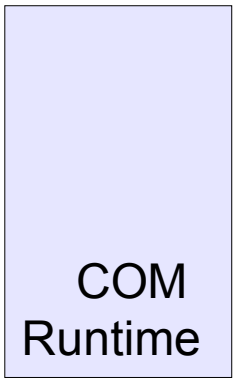
Durante la ejecución

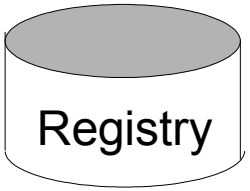


browser = {DDF7D820-8355-11CF-B357-444553540000}

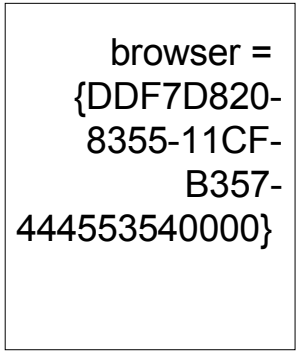
mediaplayer.ini

Durante la ejecución

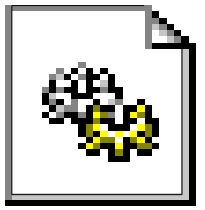




Cargado de la DLL



mediaplayer.ini



explorer.dll

Durante la ejecución

Retorno de referencia

COM Runtime

Registry

browser = {DDF7D820-8355-11CF-B357-444553540000}

mediaplayer.ini

Durante la ejecución

QueryInterface(GUID)
de iBrowser

COM
Runtime

Registry

browser =
{DDF7D820-
8355-11CF-
B357-
444553540000}

mediaplayer.ini

Durante la ejecución

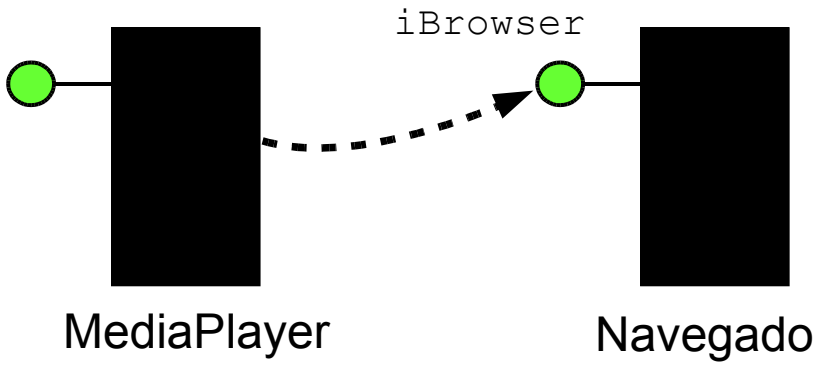
Creación de la ventana e inserción del navegador

Durante la ejecución


```
display(URL)
```

Resumiendo

- El ejemplo muestra como ensamblar una aplicación a base de componentes COM




```
browser =
{DDF7D820-
8355-11CF-
B357-
444553540000}
```

mediaplayer.ini

- La separación entre interfase e implementación permite realizar substitución
  - Basta cambiar entrada en archivo de configuración

- Como soportar la evolución de los componentes?

Evolución de componente

- **Una nueva versión del navegador es instalada**
  - Proveedor navegador != al del mediaplayer
  - Supongamos que la nueva versión introduce un cambio en la interfase

```
interface iBrowser extends iUnknown
{
     GUID={12345678-0000-0000-0000-123456789ABC}
     GraphicObject display(String URL);
}
```

V1.0

```
interface iBrowser extends iUnknown
{
     GUID={12345678-0000-0000-0000-123456789ABC}
     GraphicObject display(String URL, int zoom);
}
```

V2.0

- **Que va a pasar con los clientes?**

Durante la ejecución

QueryInterface(GUID)
de iBrowser

COM
Runtime

Registry

browser =
{DDF7D820-
8355-11CF-
B357-
444553540000}

mediaplayer.ini

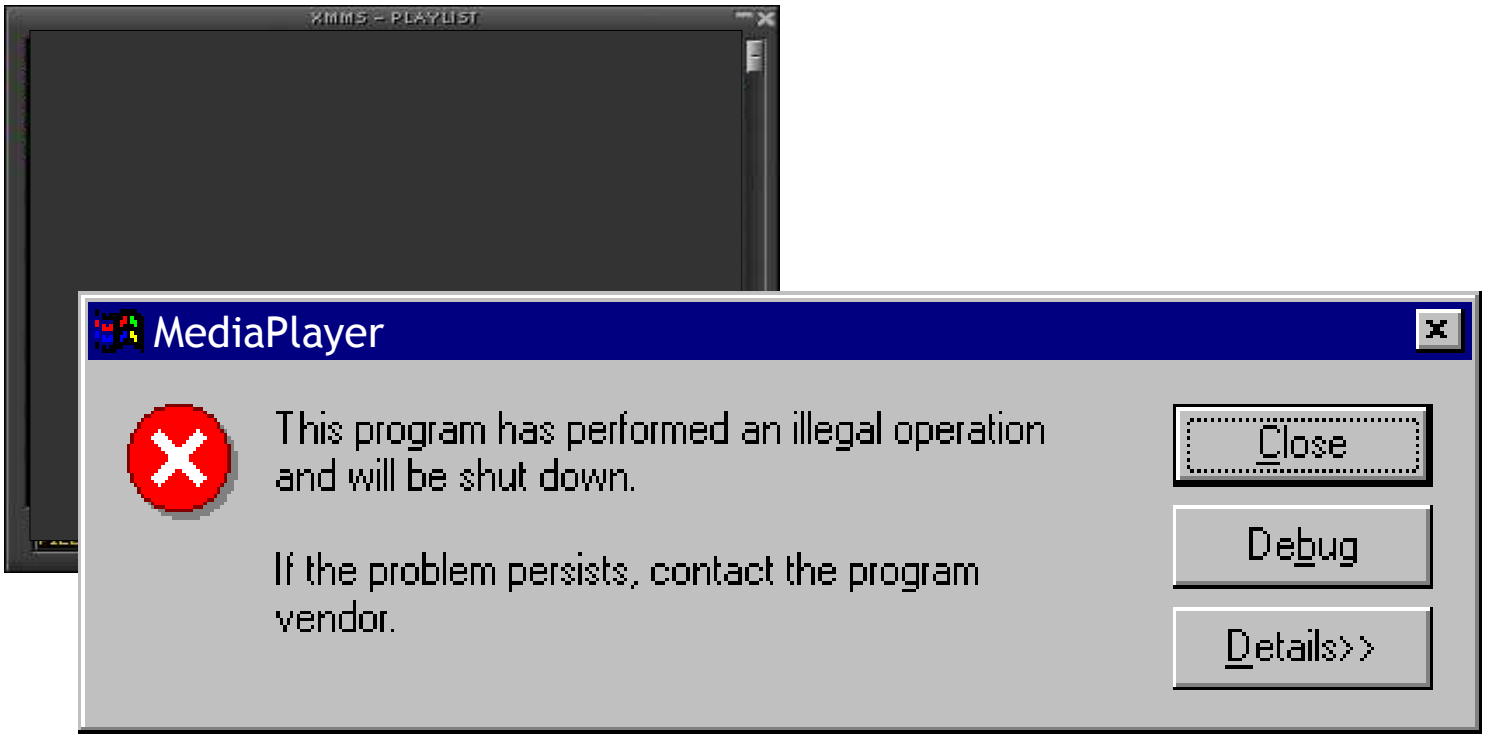
Durante la ejecución

Creación de la ventana e inserción del navegador

Durante la ejecución

- El cambio en el método de la interfase iBrowser daña a los clientes!

`display(URL)`



Evolución de componente

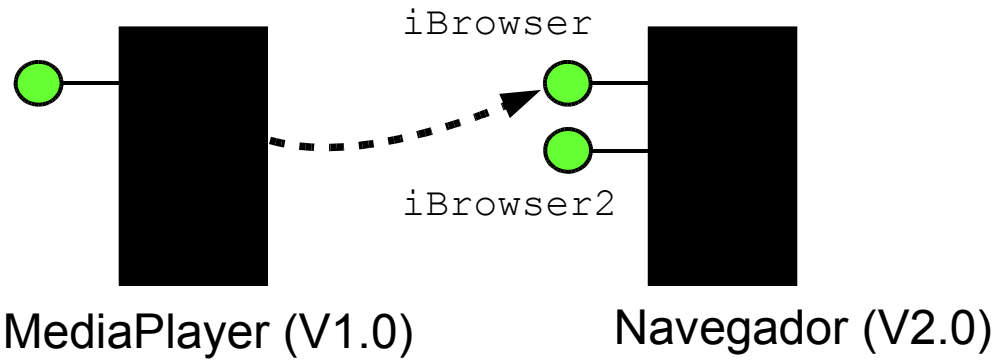
- **Solución de COM**
  - La nueva versión del navegador debe de implementar las dos interfases

```
interface iBrowser extends iUnknown
{
     GUID={12345678-0000-0000-0000-123456789ABC}
     GraphicObject display(String URL);
}

interface iBrowser2 extends iUnknown
{
     GUID={12345678-ABCD-EFGH-0000-123456789ABC}
     GraphicObject display(String URL, int zoom);
}
```

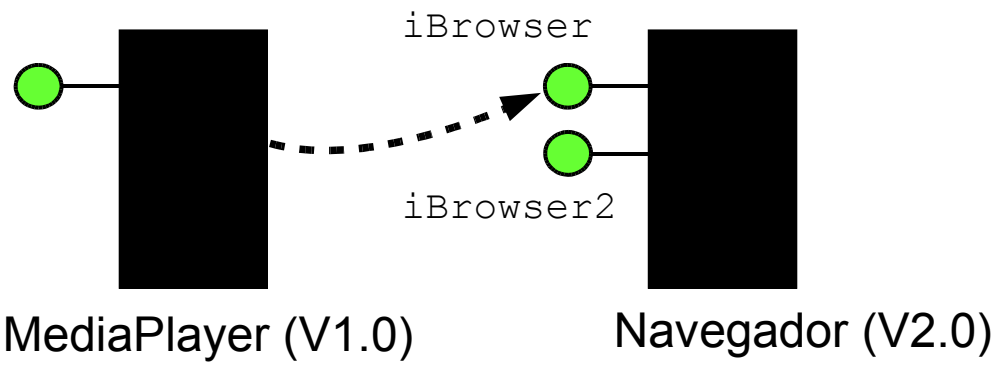
- **Así**
  - Los antiguos clientes no son dañados
  - Los nuevos clientes pueden pedir la nueva interfase

Evolución de los clientes

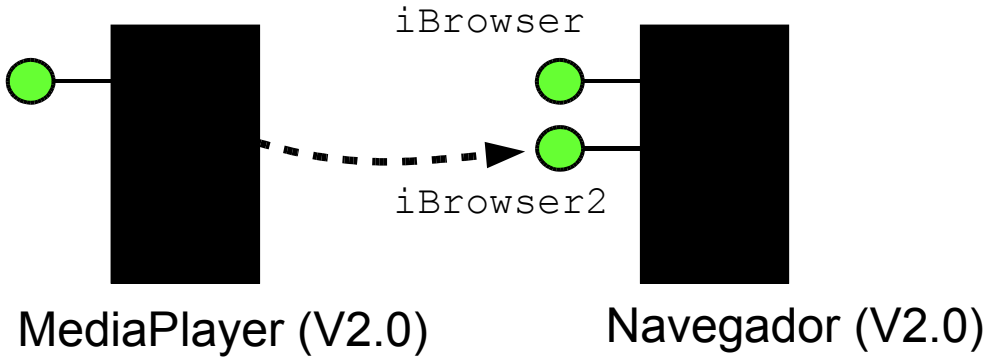


- **MediaPlayer 1.0 utiliza iBrowser**

Evolución de los clientes



- **MediaPlayer 1.0 utiliza iBrowser**



- **MediaPlayer 2.0 utiliza iBrowser2**

Conclusión COM

- **Probablemente es el modelo de componentes mas exitoso**
  - Presente en cada instalación de Windows!
  - Inicialmente para sistemas no distribuidos pero después evoluciono (DCOM, MTS, COM+)
- **Esta siendo reemplazado por .NET**
  - COM es muy complejo de programar
  - Conceptos de .NET similares a Java
  - Principios siguen vigentes
- **Fuente de inspiración para otros modelos**
  - Bonobo de GNOME

Índice

- **Introducción**
- **Conceptos principales**
- **Tecnologías existentes**
  - COM (1995)
  - JavaBeans (1997)
  - EJB (1998)
  - CCM (1999)
  - OSGi (2000)
- **Conclusiones**

JavaBeans

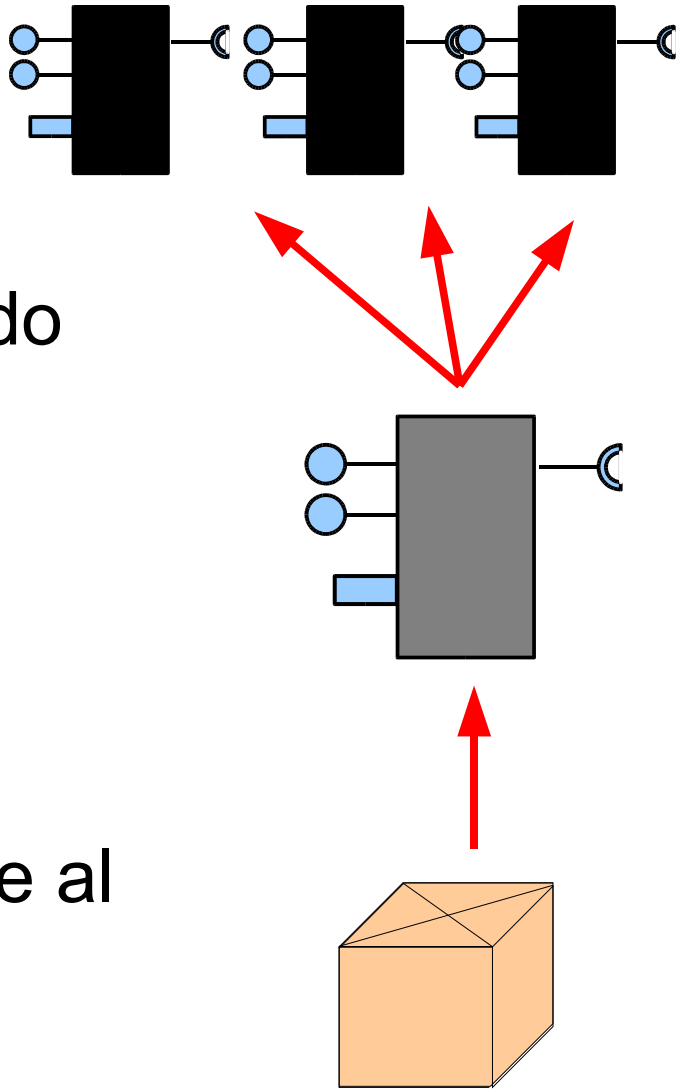
- **Sun microsystems ≈ 1997**
  - Modelo de componentes Java
  - Objetivo principal: simplificar la construcción de aplicaciones basadas en componentes de manera **visual**

- **Enfocado a aplicaciones centralizadas**
  - Lado cliente

PC Cliente

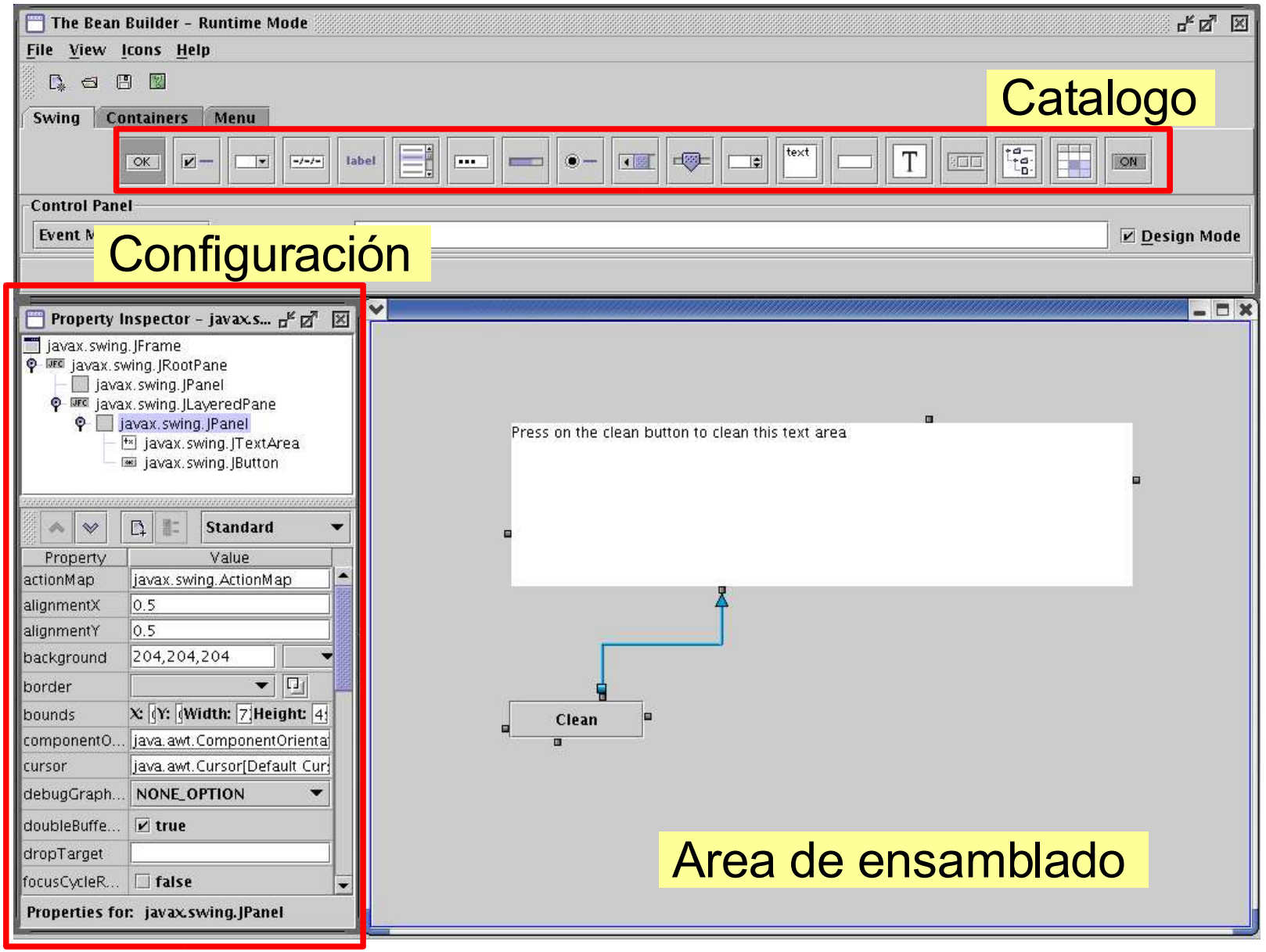
Entidades JavaBeans

- **Instancias: Objetos Java**



- **Componentes: Clases Java**
  - Siguen convenciones de nombrado
  - Proveen/requieren interfaces
  - Mandan/reciben eventos
  - Propiedades de configuración
- **Paquete: Archivos JAR**
  - Puede contener código de soporte al ensamblado
- **Entorno de ejecución: runtime Java**

Beanbuilder: Ensamblado visual



Conclusión JavaBeans

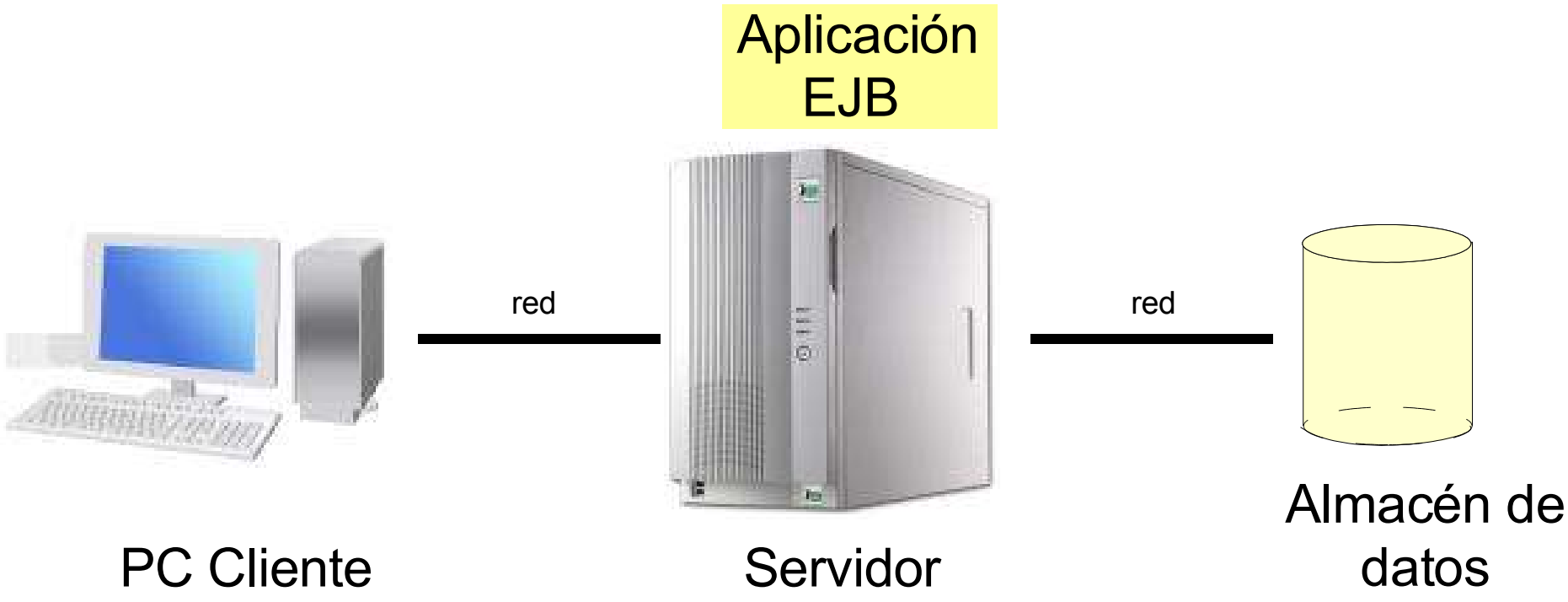
- **Modelo de componentes bastante simple**
  - Cercano a Java estándar
- **Se sigue usando para creación de interfaces gráficas**
  - Swings son componentes JavaBeans
- **Convenciones introducidas por JavaBeans se usan hoy en día para aspectos de configuración**
  - get/set
  - add/remove

Índice

- **Introducción**
- **Conceptos principales**
- **Tecnologías existentes**
  - COM (1995)
  - JavaBeans (1997)
  - EJB (1998)
  - CCM (1999)
  - OSGi (2000)
- **Conclusiones**

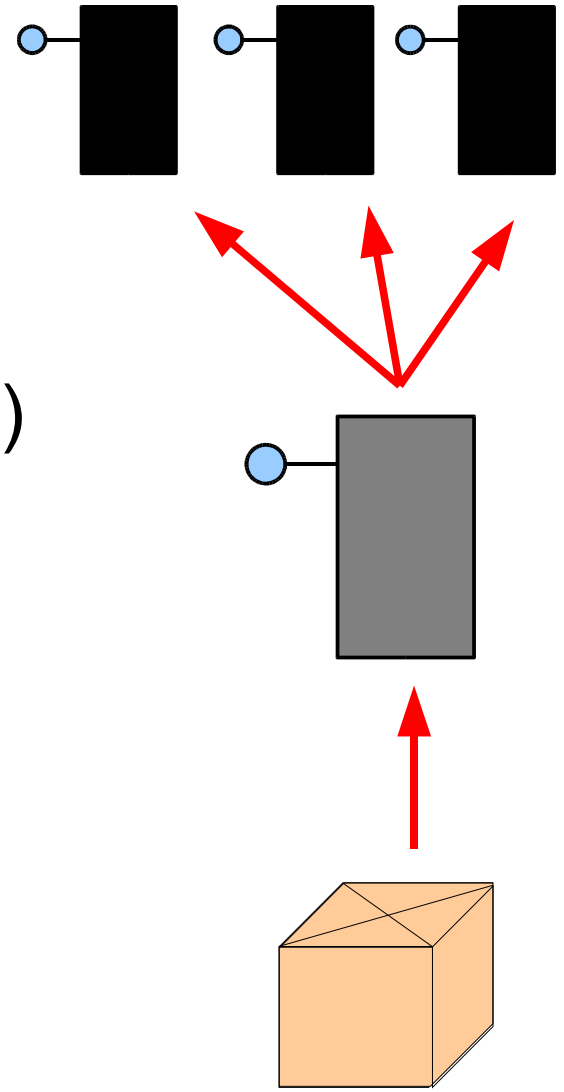
Enterprise JavaBeans

- **Sun microsystems ≈ 1998**
  - Modelo de componentes Java orientado a la construcción de aplicaciones 3 tercios (“3-tier”). EJB se orienta particularmente al tercio medio, es decir el servidor aplicativo.



Entidades EJB

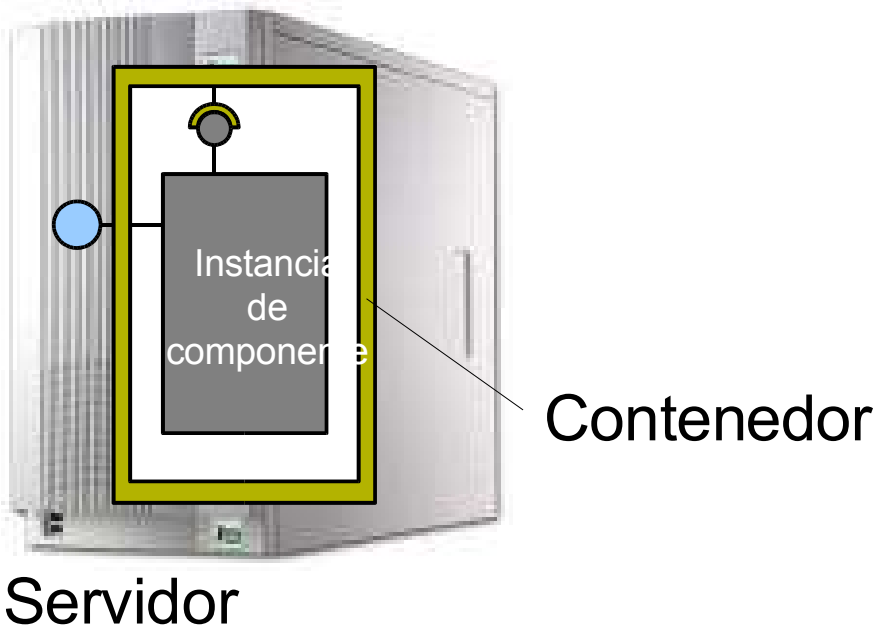
- **Instancias: Objetos Java**
  - Persistentes o no
- **Componentes: Beans**
  - Proveen una sola interfase (*remote*)
  - 3 tipos
    - Entity
    - Session
    - Message
- **Paquete: Archivos EAR**
  - Descriptor de despliegue permite configurar de forma declarativa



3 Tipos de Beans

- **Entity beans (datos)**
  - Identificados de manera única
  - Persistentes
  - Por ejemplo: una cuenta de banco
- **Session beans (lógica de aplicación)**
  - Único medio de interacción con clientes
  - Viven lo que dura una interacción con un cliente
  - Ejemplo: carrito de compras
- **Message beans (lógica de aplicación)**
  - Similares a session beans
  - Comunicación por mensajes

Contenedor EJB



- **Parte del entorno de ejecución. Se encarga de:**
  - Manejo del ciclo de vida (activación / pasivación)
  - Manejo de aspectos no funcionales
    - Distribución, persistencia, seguridad, transacciones
    - Mediante intercepción

Conclusión EJB

- **Modelo de componentes lado servidor**
  - Ampliamente usado en la actualidad
  - Sin embargo su utilización es compleja
- **Complementado por otras tecnologías**
  - Servlets y JSP: generación de paginas dinámicas
  - JavaServer Faces: creación de interfaces usuario
  - Mbeans: administración
  - etc...

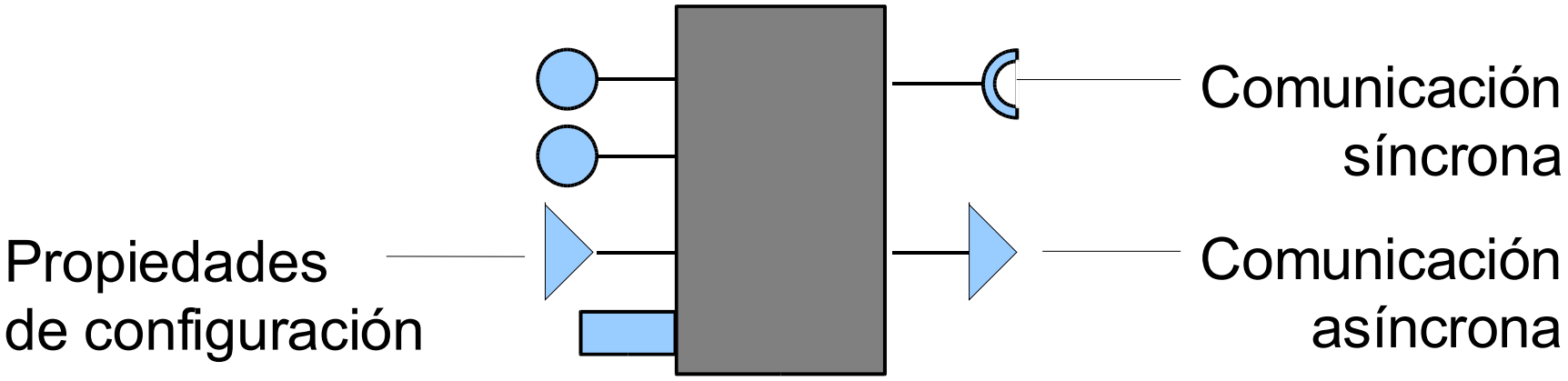
Índice

- **Introducción**
- **Conceptos principales**
- **Tecnologías existentes**
  - COM (1995)
  - JavaBeans (1997)
  - EJB (1998)
  - CCM (1999)
  - OSGi (2000)
- **Conclusiones**

Corba Component Model (CCM)

- **Object Management Group (OMG) ≈ 1999**
  - Modelo de componentes construido por encima de CORBA
    - CORBA: modelo de objetos distribuidos independientes del lenguaje de programación + infraestructura que provee un conjunto de servicios
  - Objetivo principal de CCM es facilitar el despliegue de aplicaciones distribuidas
- **Una especificación muy compleja**
  - No hubo una implementación completa por varios años!

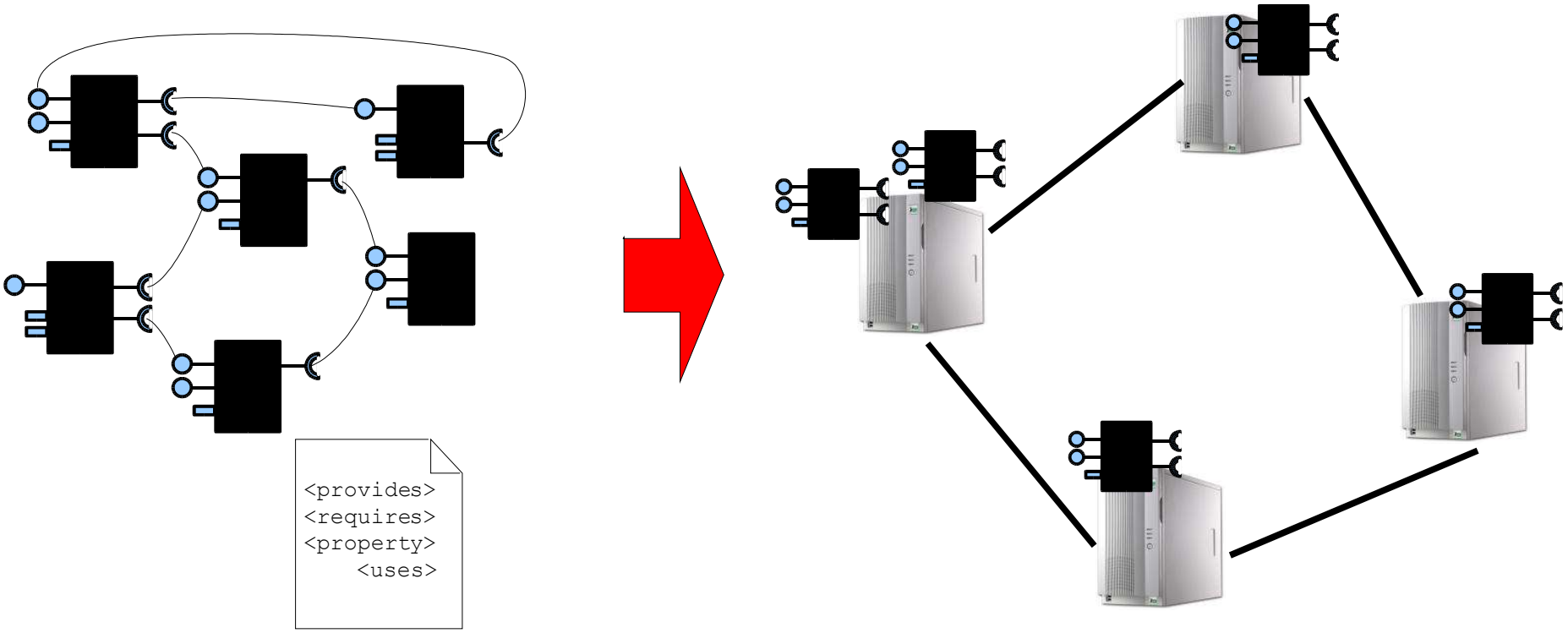
CCM: Componentes



- **Componente CCM**
  - Varios tipos: service, session, process, entity
- **Descripción / implementación**
  - Descrito en un lenguaje abstracto (IDL3)
  - Puede ser implementado en distintos lenguajes

CCM: Assembly

- **Que es?**
  - El assembly describe de forma declarativa una composición de instancias de componentes CCM.
  - Es usada para desplegar y lanzar una aplicación



Conclusiones CCM

- **Ideas buenas**
  - Descripción abstracta de un componente
  - Implementación multi-lenguaje
  - Unico que provee un mecanismo que permite describir una composición
  - 'engloba' a los EJB

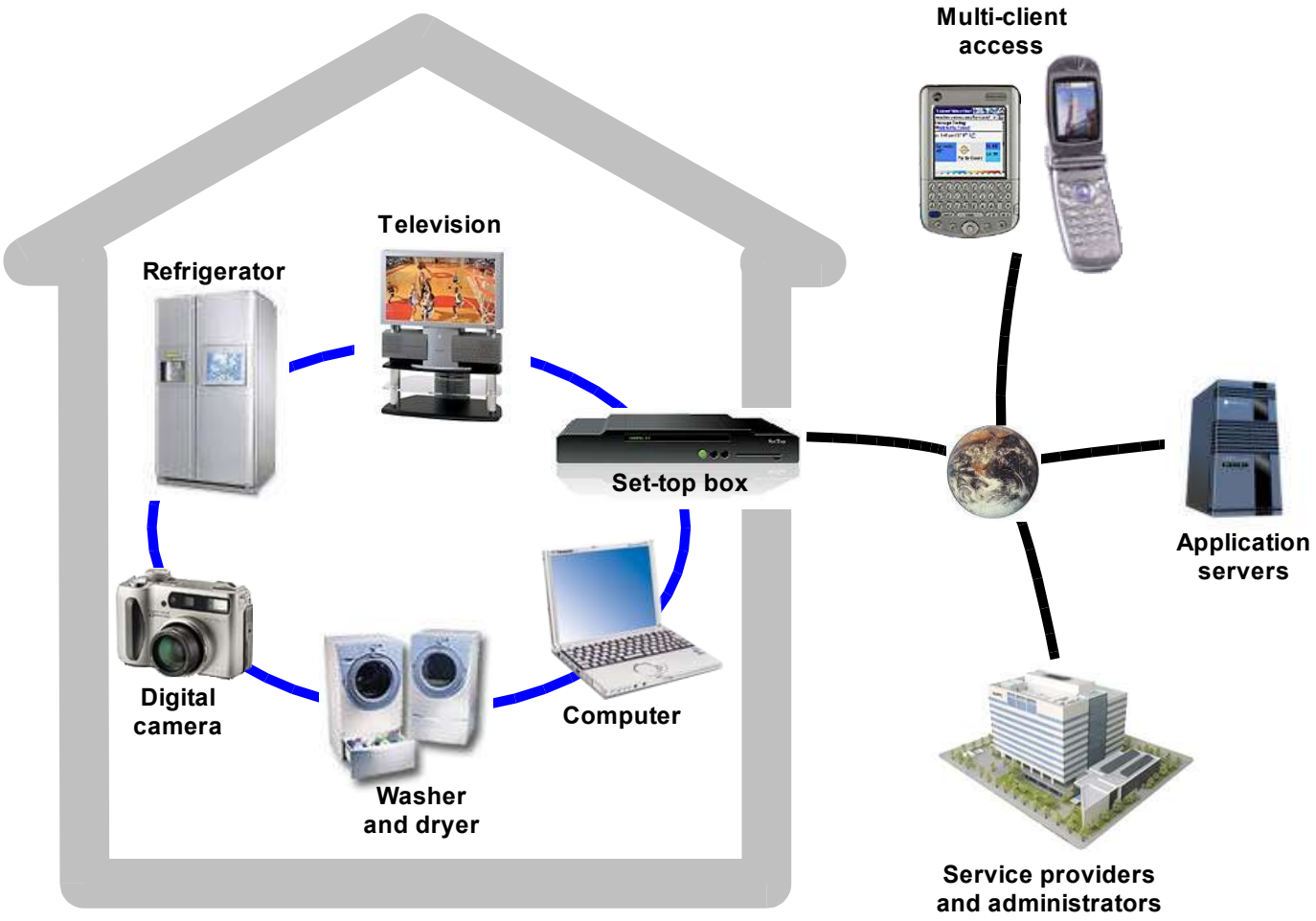
- **Hace 5 años generó muchas expectativas...**
  - Sin embargo, la falta de una implementación completa y la complejidad de la especificación no han facilitado su adopción
  - Futuro no muy claro...

Índice

- **Introducción**
- **Conceptos principales**
- **Tecnologías existentes**
  - COM (1995)
  - JavaBeans (1997)
  - EJB (1998)
  - CCM (1999)
  - OSGi (2000)
- **Conclusiones**

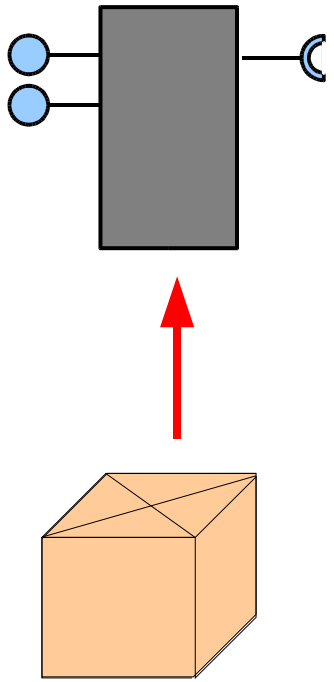
OSGi

- **Open Services Gateway Initiative ≈ 2000**
  - Objetivo original: permitir la interacción de aparatos dentro de una red residencial



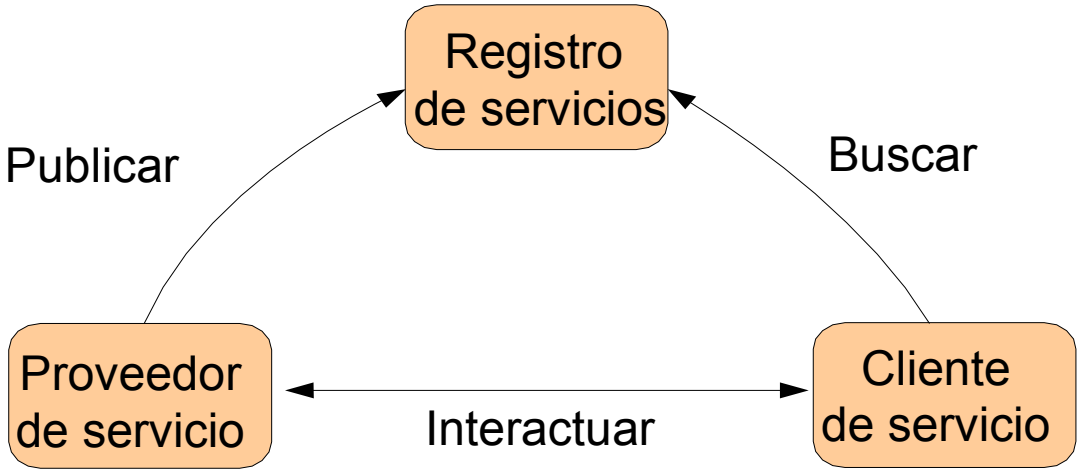
Entidades OSGi

- **Instancias: no hay instancias múltiples**
- **Componentes: Bundle lógico**
  - Provee y requiere servicios
- **Paquete: Bundle físico**
  - Archivos JAR
- **Entorno de ejecución**
  - Framework OSGi

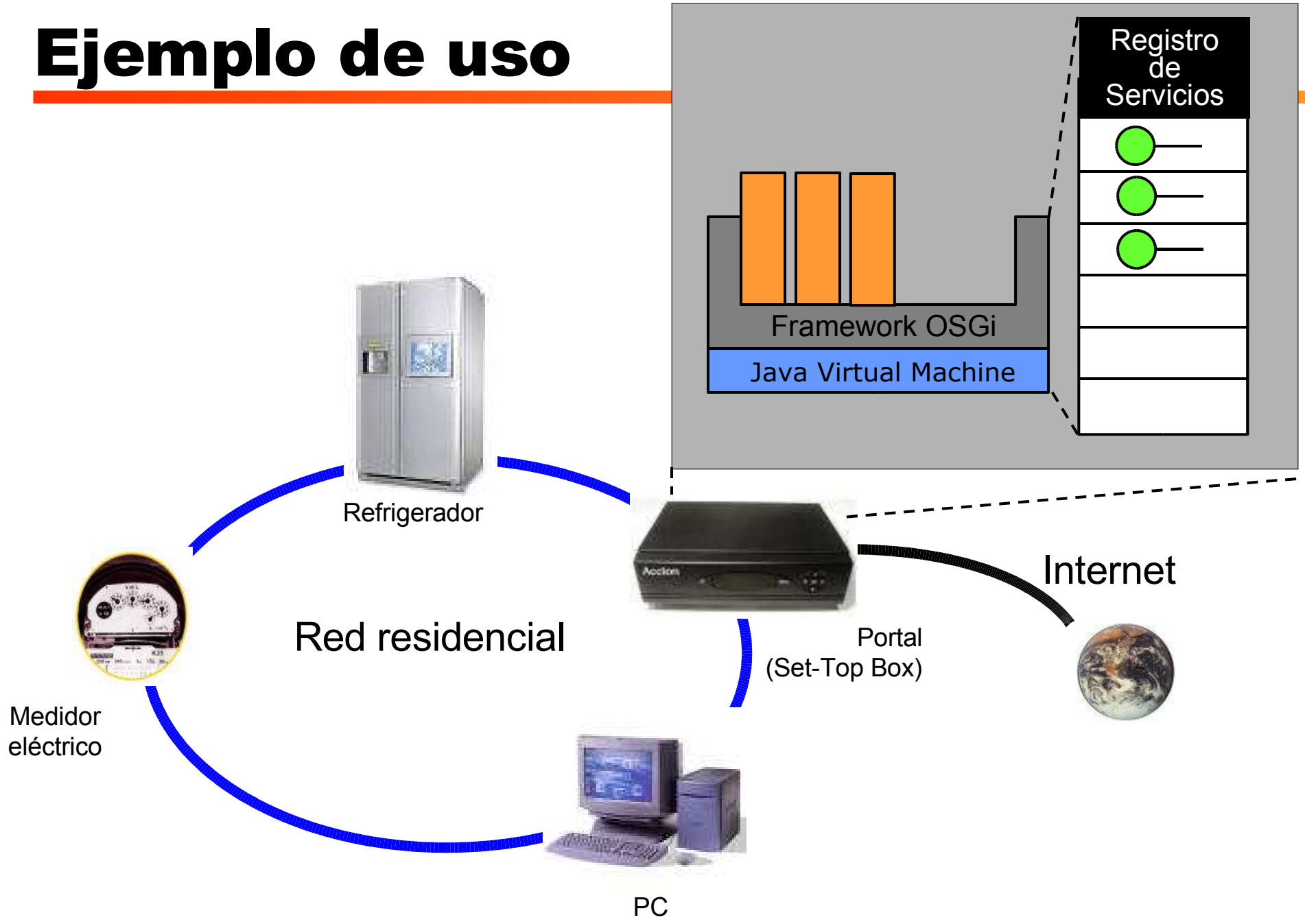


Entorno de ejecución

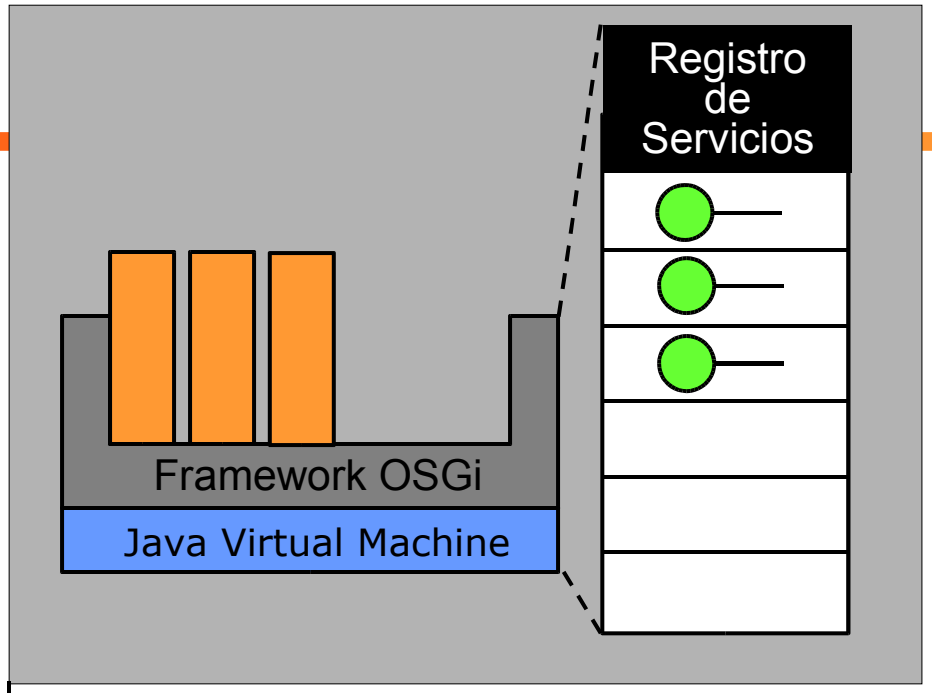
- **Soporte para el despliegue continuo**
  - En todo momento se pueden instalar, activar, actualizar, desactivar y retirar paquetes de componentes
- **Soporte para el *trading***
  - Los componentes lógicos pueden publicar y buscar sus servicios en un registro



Ejemplo de uso



Ejemplo de uso



Refrigerador

Portal
(Set-Top Box)

Internet

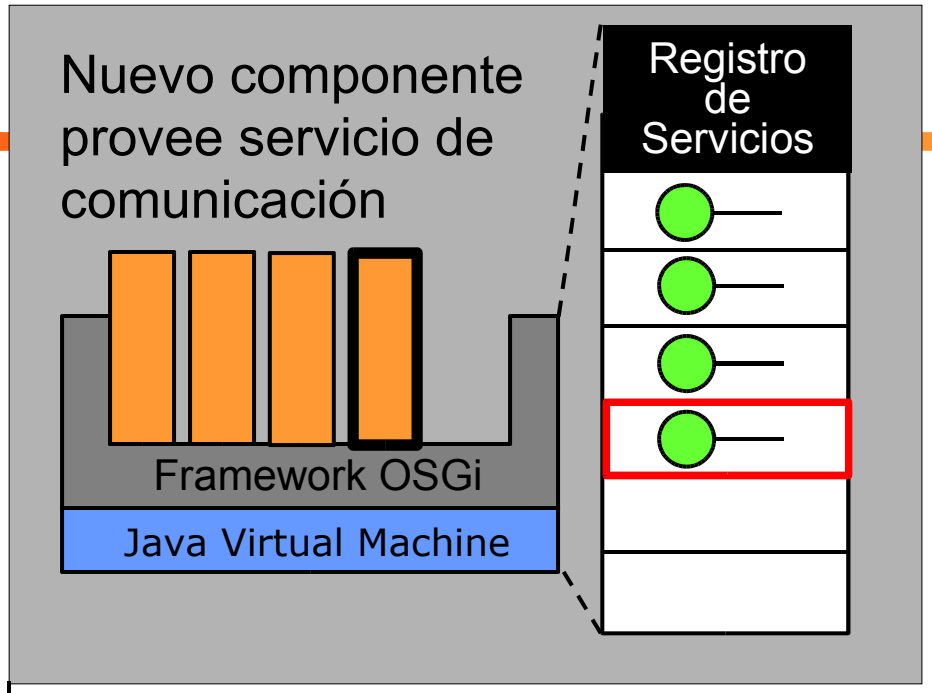
Medidor
eléctrico

Red residencial

Teléfono

PC

Ejemplo de uso



Refrigerador

Portal
(Set-Top Box)

Internet

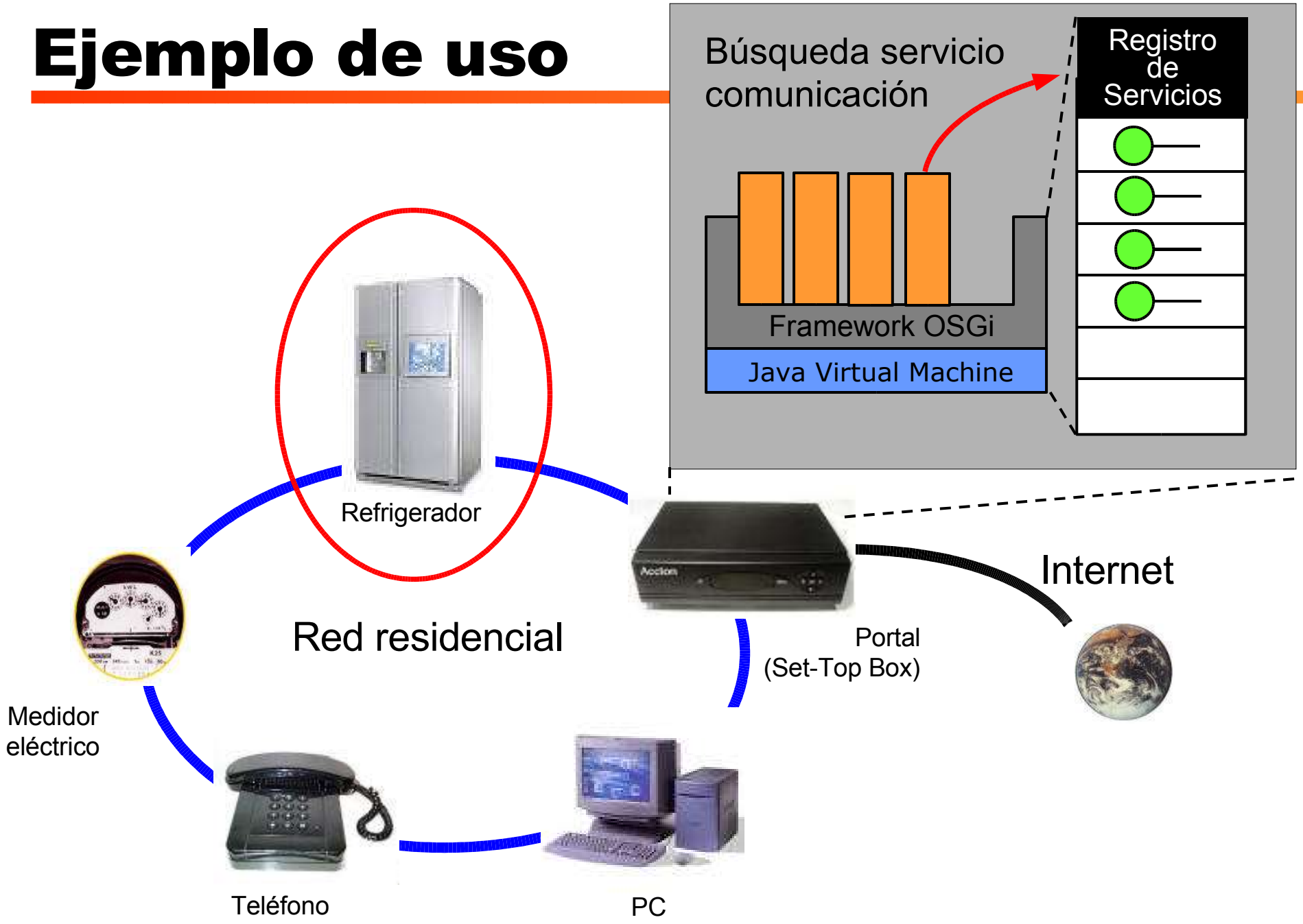
Medidor
eléctrico

Red residencial

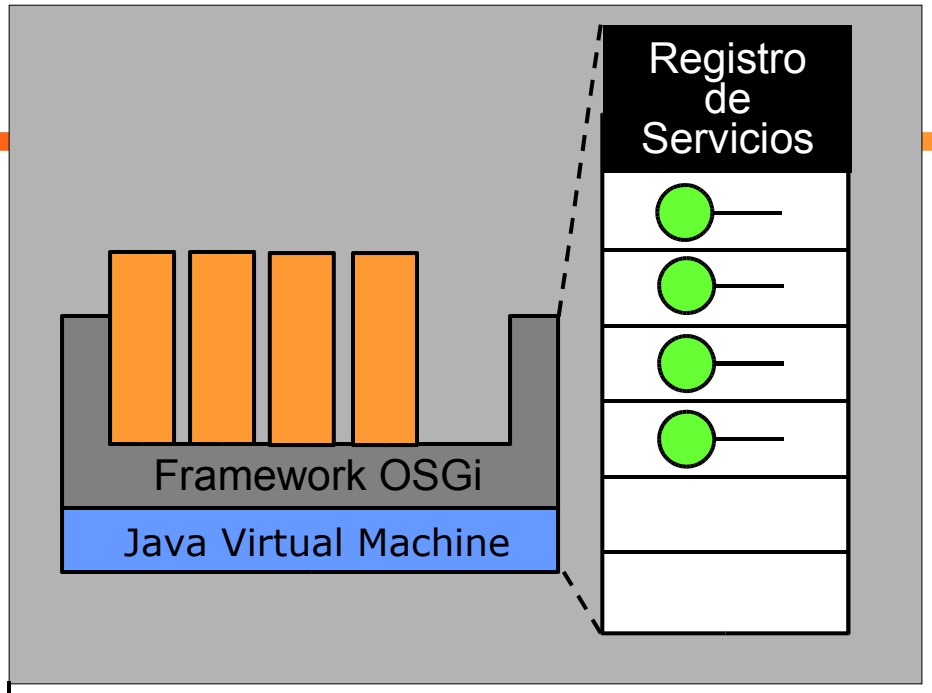
Teléfono

PC

Ejemplo de uso



Ejemplo de uso



Refrigerador

Portal
(Set-Top Box)

Internet

Red residencial

Medidor
eléctrico

Teléfono

PC

“Falta leche”

Conclusiones OSGi

- **Entorno sumamente dinámico**
  - Componentes pueden ser instalados o retirados mientras se ejecuta la aplicación
- **OSGi presente y futuro**
  - OSGi utilizado en otras áreas: teléfonos celulares, automóviles
  - OSGi también se esta utilizando como infraestructura para el IDE Eclipse (desde la 3.0)
  - En general OSGi se puede emplear para desplegar aplicaciones Java

Índice

- **Introducción**
- **Conceptos principales**
- **Tecnologías existentes**
- **Conclusiones**
  - Síntesis
  - El futuro
  - Conclusión

Síntesis

- **Hoy en día los componentes están en todo el software que conocemos**
  - En Windows / Linux del lado 'cliente'
  - En las aplicaciones del lado 'servidor', por ejemplo en el comercio electrónico
- **Están apareciendo en todos lados en donde hay software**
  - Teléfonos celulares
  - Control de automóviles
  - Redes residenciales
  - Etc..

Síntesis

- **La ingeniería de software basada en componentes es una evolución de metodologías anteriores.**
  - Incorpora principios de modularidad, programación orientada a objetos, etc...
- **Hace énfasis en la composición por terceros**
  - Expresar las dependencias de forma explicita es una necesidad
  - Permite la reutilización
- **Sin embargo no existe una metodología de desarrollo basada en componentes**

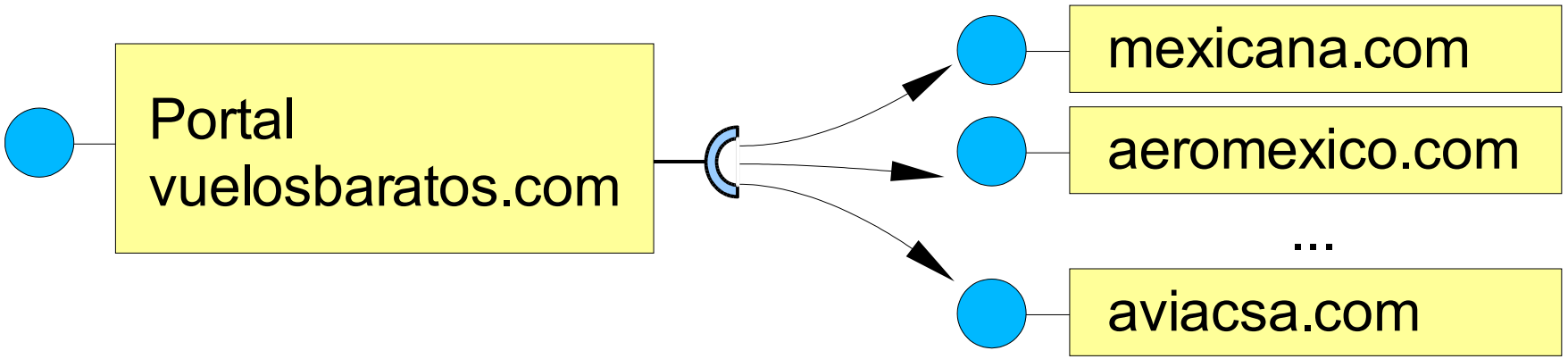
La reutilización, una realidad?

- **Cada vez hay más componentes reutilizables**
  - Internet juega un papel clave para la existencia de catálogos y mercados de componentes
- **Sin embargo hay problemas difíciles de resolver**
  - Como saber si un componente hace lo que se supone que debe hacer? Como saber si no contiene código maligno?
  - Como evolucionar un componente sin afectar a los clientes?

Índice

El futuro

- **Servicios Web : la reutilización a un nivel mayor de granularidad**
  - Por ejemplo toda una aplicación de consulta de vuelos de una linea aérea puede ser incorporada dentro de una aplicación que busca el mejor precio de un boleto de avión



- **Servicios Web y componentes son tecnologías complementarias**

El futuro

- **Proliferación de tecnologías: un problema**
  - Cuesta mucho trabajo dominar un modelo de componentes
  - Muchos detalles técnicos ocultan los conceptos
- **Una propuesta de solución es MDA (model driven architecture)**
  - Una aplicación se define como un modelo usando UML
  - El modelo se mapea a distintas tecnologías de componentes
    - La tecnología de implementación es secundaria

Índice

- **Introducción**
- **Conceptos principales**
- **Tecnologías existentes**
- **Conclusiones**
  - Síntesis
  - El futuro
  - Conclusión

Componentes: llegaron para quedarse

- **Los componentes son una evolución no una revolución**
- **Obligan a seguir buenas practicas de desarrollo que existen desde hace mucho tiempo...**
  - Encapsulamiento
  - Modularidad
  - Despliegue
- **Hoy en día es difícil hacer software sin tomarlos en cuenta**

Preguntas?