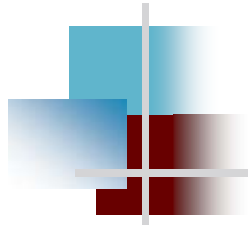


Desarrollo basado en Componentes y Servicios

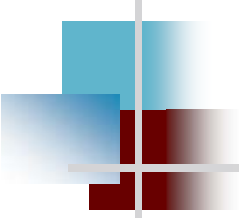
Grado en Ingeniería Informática

Miguel A. Laguna



Objetivos del curso

- Aprender:
 - Fundamentos del software basado en componentes
 - Modelos de Componentes
 - Componentes ejecutables y Frameworks de componentes
 - Servicios Web
 - Características de la Ingeniería del Software basada Componentes y/o Servicios
- Ser capaz de:
 - Diseñar e implementar un sistema utilizando las herramientas, técnicas y procedimientos mostrados en el curso



Desarrollo basado en Componentes y Servicios

- Conceptos de Desarrollo Basado en Componentes
- Modelos de Componentes
- Tecnologías y marcos de trabajo para desarrollo de sistemas de componentes distribuidos.
- Ingeniería del Software basada en Componentes
- Tecnología de Servicios Web
- Desarrollo de Software orientado a Servicios



Bibliografía general: Conceptos

- 📖 Sommerville, I. "Ingeniería del software" Pearson, 2005 (7ª ed.)
- 📖 Andy Ju An Wang, Kai Qian. Component-oriented programming. John Wiley & Sons, 2005

Lecturas complementarias

- 📖 Arlow, Jim, Neustadt, Ila. "UML 2", Anaya Multimedia, 2006
- 📖 Clemens Szyperski "Component Software - Beyond Object-Oriented Programming" – Second Edition, Addison-Wesley / ACM Press, 2002



Bibliografía general: Tecnología de componentes

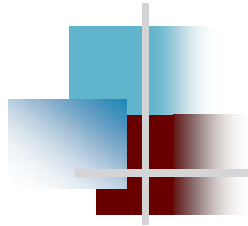
EJB/JEE

📖 D. Panda, R. Rahman, D. Lane. EJB3 in Action. 2aEd Manning. 2014.
ISBN: 1-933988-34-7.

📖 Oracle/Sun, The JEE Tutorial

.NET

📖 Juval Lowy, Programming .NET Components, 2nd Edition O'Reilly.
2005. ISBN-13: 978-0596102074



Bibliografía general: Diseño

Generales

- 📖 John Cheesman & John Daniels. UML components: a simple process for specifying component-based software. Addison-Wesley, 2000
- 📖 Martin Fowler. Patterns of Enterprise Application Architecture. Addison Wesley, 2003.

Específicas EJB/.NET

- 📖 Derek C. Ashmore. The JEE Architect's Handbook. DVT Press. ISBN: 2004
- 📖 Murat Yener, Alex Theedom, Professional Java EE Design Patterns, ISBN: 978-1-118-84341-3, 264 pages, Wiley, 2015
- 📖 Microsoft patterns & practices <http://msdn.microsoft.com>
 - 📖 Microsoft. Enterprise Solution Patterns Using Microsoft .NET
 - 📖 Microsoft. Application Architecture Guide, 2nd Edition. ISBN: 9780735627109.



Bibliografía general: Servicios Web

SOAP/SOA

- 📖 Michael Papazoglou, "Web Services and SOA: Principles and Technology", 2ª ed. Pearson, 2012

REST

- 📖 Leonard Richardson, Sam Ruby. RESTful Web Services, Web services for the real world. O'Reilly, 2007.

DISEÑO

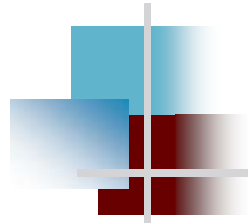
- 📖 Robert Daigneau. "Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services", Addison 2012



1. Introducción:

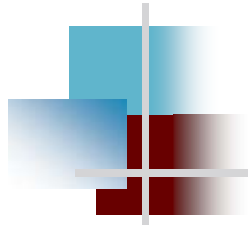
Conceptos de Desarrollo Basado en Componentes

Miguel A. Laguna



Desarrollo del tema

- 1.0. La visión general
- 1.1. Reutilización de software
 - Enfoques
- 1.2. Beneficios de los componentes
 - Evolución de la tecnología de componentes
 - Mercado y estándares
- 1.3. Fundamentos y definiciones
 - Componentes frente a objetos: composición frente a herencia
- 1.4. Interfaces y Contratos
- 1.5. CBSE: Ingeniería de Software Basada en Componentes



Bibliografía

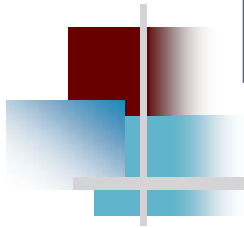
- 📖 Sommerville, I. "Ingeniería del software" Pearson, 2005 (7ª ed.)

Lecturas complementarias

- 📖 Clemens Szyperski "Component Software - Beyond Object-Oriented Programming" – Second Edition, Addison-Wesley / ACM Press, 2002
- 📖 Ivica Crnkovic, Magnus Larsson, Building Reliable Component-based Systems, Artech House; 2002

1.0

La visión

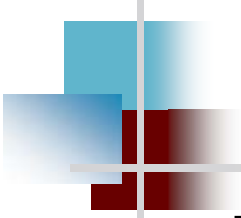


Desarrollo basado en componentes

- Gestión de la complejidad: divide y vencerás
- Bien conocido en otras disciplinas
 - Ingeniería Mecánica
 - Ingeniería Eléctrica
 - Arquitectura de Computadoras
 - Arquitectura



Desarrollo basado en componentes

- 
- Idea principal: construir un sistema conectando partes más pequeñas, independientes, ya existentes, (quizás) provenientes de terceros.
 - Concepto “Commercial Off-The-Shelf” (COTS)
 - Reducir todo lo posible la creación de código nuevo
 - Los componentes están bien definidos en otras ingenierías pero no en ingeniería de software, debido a la naturaleza del software



La visión del Desarrollo basado en Componentes

- Entidades básicas: Componentes
 - Cajas negras que encapsulan cierta funcionalidad y que son diseñadas para un “Mercado Global de Componentes” sin saber quién las utilizará, ni cómo, ni cuándo.
 - Pueden ser componentes comerciales y/o gratuitos.
- Aplicar la idea de la industrialización al desarrollo de software.
 - El programador dispone de una “paleta” de componentes y trabaja como en una cadena de montaje, escribiendo solo el código de interconexión



Necesidad del Desarrollo de Componentes

- Mercado global de componentes para los desarrolladores de aplicaciones que necesitan reutilizar componentes probados para construir sus aplicaciones de forma más rápida y robusta
 - (o que quieren añadir funcionalidad desarrollada por terceros).
- Component source:
 - <http://www.componentsource.com/index.html>



¿Variante de la programación orientada a objetos?

- La Programación orientada a objetos clásica se centra en el código fuente (las clases) que serán después compiladas en un programa binario ejecutable.
- Si alguna clase sufre cambios:
 - Re-compilación masiva de la aplicación completa
 - Realizar nuevamente las pruebas
 - Re-implementación posiblemente de otras clases.



Componentes: Diferencias con OO

- Se centra en módulos intercambiables fabricados independientemente y no se requiere conocer su implementación interna.
- Si es necesario modificar un componente:
 - Los cambios solo en ese componente.
 - No existe la necesidad de re-compilación o re-implementación.
 - Pueden ser actualizados aunque el sistema esté ejecutándose, si el componente no se encuentra en uso.
- Una aplicación orientada a componentes es fácil de extender.
 - Cuando se implementan nuevos requisitos, se pueden añadir nuevos componentes, sin tocar los existentes.
- Se reduce el coste de mantenimiento.



Una Aclaración importante

- Computación distribuida: Comunicación entre componentes localizados en diferentes máquinas.
distribuido != componentes
- Los componentes pueden ejecutarse en modo local o remoto (usando un protocolo común)
- Comunicación entre componentes remotos:
 - Mensajes (sockets), RPC
 - Objetos distribuidos: CORBA Middleware: servicios mínimos para localizar objetos, gestionar llamadas entre componentes distribuidos, etc.
 - JEE, .NET Remoting se ejecutan sobre ese middleware



1.1. Reutilización



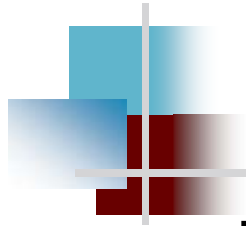
Desarrollo a medida y Software Estándar

- Dos tipos fundamentales de desarrollo tradicional de software:
 - Un proyecto **desarrollado en su totalidad** línea por línea con solo la ayuda de herramientas de programación y bibliotecas.
 - Se compra **software estándar y se parametriza** para proporcionar una solución que esté cerca de lo que necesita el cliente.



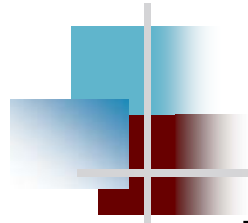
Desarrollo a medida

- La producción este tipo de software es **muy costosa** (mantenimiento!).
- Suponen **un riesgo**
 - La mayoría de los proyectos grandes fallan parcialmente o incluso totalmente
 - Problemas de interoperabilidad con otros sistemas
- En un mundo de rápidos y continuos cambios en los requisitos, este tipo de software corre el riesgo de convertirse en obsoleto antes de ser productivo



Software estándar

- Disminuye el riesgo.
 - El vendedor debe disminuir los problemas del mantenimiento, de la evolución del producto, y de la interoperabilidad
- Requiere parametrización y configuración detallada entre versiones (inevitable por los cambios)
- Implica una **reorganización** del proceso del negocio
- Debido a que no está bajo control local, **no se adapta rápidamente a cambios** en los requisitos que surgirán



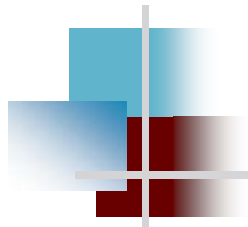
Reutilización del software

- El reto es construir software rápidamente adaptable a los cambios y gestionar la cada vez mayor complejidad
- Mecanismos:
 - Encapsulado, módulos
 - Uso de estándares
 - Interacción entre aplicaciones o partes de ellas (Suite Office)
 - Desarrollo de un mercado (industrial)
 - En general: Reutilización del software



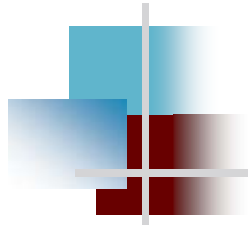
Reutilización del software: Ventajas y desventajas

Ventajas	Desventajas (y frenos)
Costes totales de desarrollo se reducen	Incremento inicial de costes de creación
Incremento de la fiabilidad	Creación y mantenimiento de la biblioteca de módulos o componentes
Reducción del riesgo del proceso	Búsqueda, comprensión y adaptación de módulos o componentes
Desarrollo acelerado	Faltan herramientas
Cumplimiento de estándares	
Uso efectivo de especialistas	Faltan especialistas



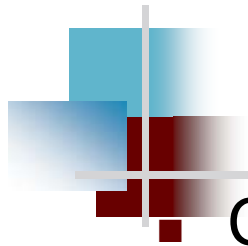
Formas de reutilización





Niveles de Reutilización

- Artefactos de análisis y diseño
 - Patrones de diseño: reutilización de experiencia, adaptación a cada caso concreto
 - Frameworks: arquitectura común
 - Familias de aplicaciones, Líneas de productos software: arquitectura común y variable
- Código fuente
 - Módulos, clases y herencia (OOP), separación de aspectos (AOP)
 - Frameworks: esqueleto de código que hay que completar
 - Líneas de productos software con código fuente (común y opcional)



Niveles de Reutilización

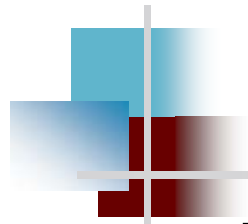
■ Código ejecutable

- Bibliotecas pre-compiladas (paquete para gráficos) y compartidas (enlace en tiempo de ejecución: DLLs)
- Aplicaciones completas que interaccionan (filtros UNIX)
- Componentes básicos (por ejemplo, paletas para construcción de aplicaciones de escritorio: controles .NET, Java beans...)
- Componentes distribuidos estandarizados: EJB, .NET Remote
- Servicios Web
- COTS, ERP, componentes configurables
- Líneas de productos software implementadas con DSLs, Generadores de aplicaciones,...



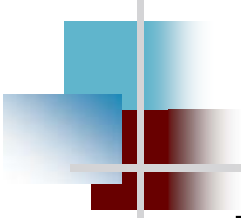
Formas de reutilización

Reutilización de código fuente	Reutilización de componentes ejecutables
Bibliotecas de módulos	Bibliotecas de módulos ejecutables
Bibliotecas de clases	Envoltorios (Wrapping) de sistemas legados
Patrones de diseño	COTS, Aplicaciones Configurables (ERP)
Frameworks (Marcos de trabajo) de aplicaciones (GUI, DB)	Frameworks de componentes, DBC
Desarrollo del software orientado a aspectos	Sistemas orientados a servicios
Líneas de productos software	Lenguajes específicos de dominio
	Generadores de aplicaciones, MDD



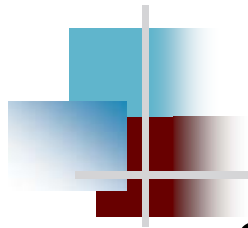
Frameworks

- Un marco de trabajo (o Framework) es un diseño de un sistema incompleto formado por una colección de clases concretas y abstractas y la interfaz entre ellas
- Suelen estar diseñados con patrones
- Complejos y difíciles de aprender pero muy efectivos
 - construcción de interfaces usuario, GLADE, Swing,...
 - Conexión con BD: Hibernate
- También relacionados con formas de reutilización basados en componentes:
 - Infraestructura de sistemas: comunicaciones, integración de middleware, modelos de componentes estandarizados como COM+, .NET, JEE, ...



Componentes “commercial-off-the-shelf” (COTS)

- Un sistema de software que puede ser adaptado para diferentes clientes sin cambiar el código fuente del sistema.
- Los sistemas COTS tienen características genéricas y se puede utilizar en diferentes entornos.
 - Mecanismos de configuración que permiten afinar la funcionalidad del sistema para adaptarse a las necesidades específicas de los clientes.
 - Por ejemplo, en un sistema de registro de pacientes de un hospital, pueden definirse formularios de ingreso y alta para diferentes tipos de pacientes a partir de uno común

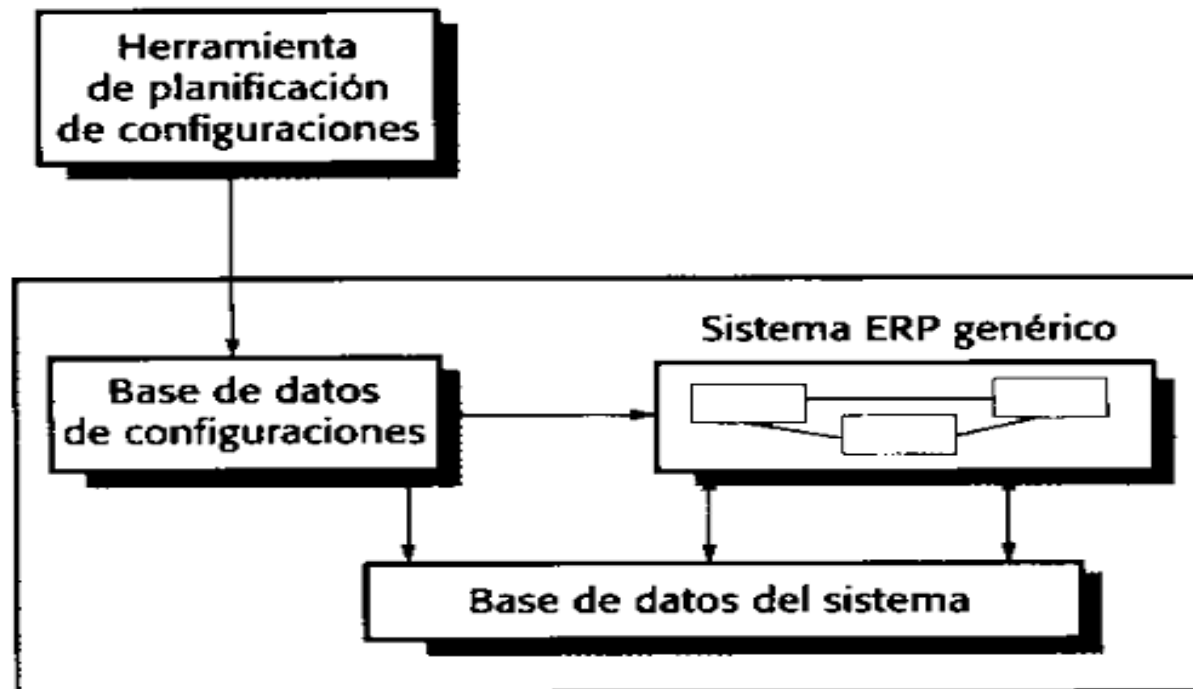


Tipos de COTS

- Soluciones COTS
 - Sistemas genéricos diseñados para un tipo de negocio en particular o actividad empresarial
 - Solución COTS para dentistas (maneja citas, registros dentales, datos de paciente), autoescuelas, etc.
- COTS específicos de un dominio (transversal)
 - Proporcionan una funcionalidad que es probable que requieran muchos usuarios potenciales
 - Gestión de documentos, creación de formularios, Aplicación de contabilidad general

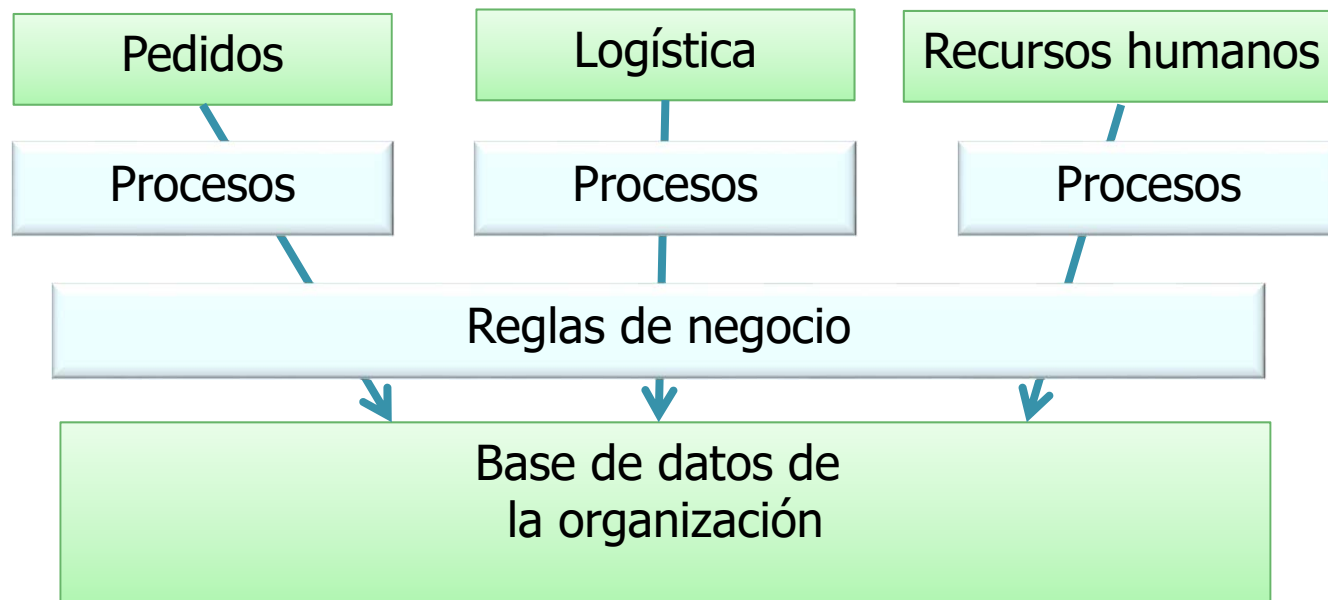
COTS a gran escala: ERP

- Enterprise Resource Planning (SAP, Navision,..)
- Muy utilizados en grandes empresas
- Los sistemas de planificación de recursos Soportan los procesos de negocio comunes, como pedidos, facturación, fabricación...



Enterprise Resource Planning

- Se crea una instancia de un sistema ERP mediante la configuración de un sistema genérico con información sobre los **procesos y las reglas** de negocio del cliente



Combinación de estrategias

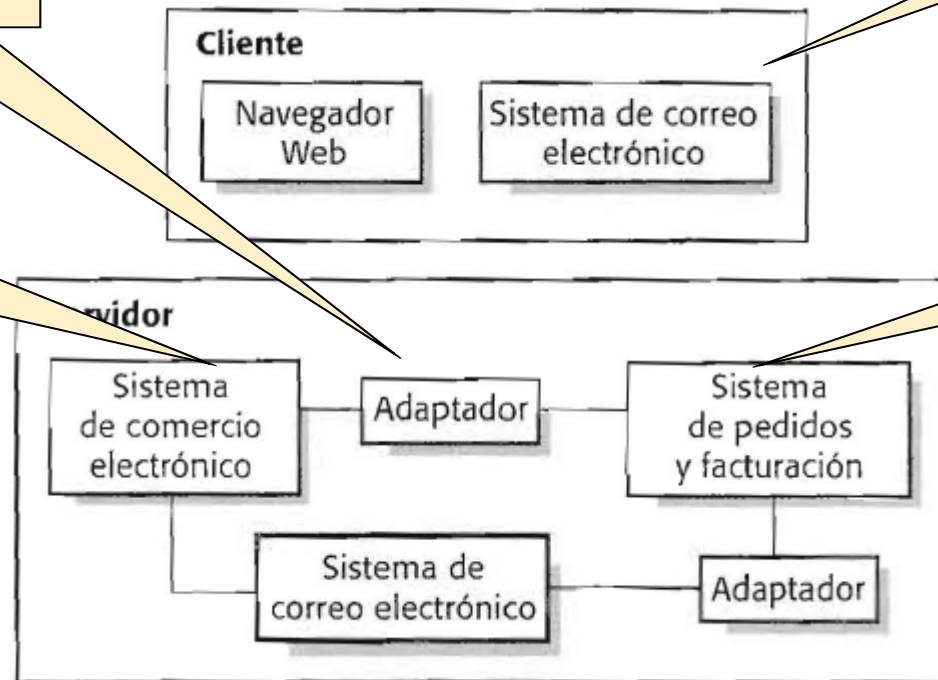
¿Reconoce alguna posibilidad de reutilización?

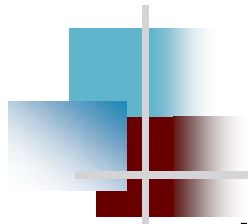
Envoltorio

COTS

COTS o
Framework

Sistema
legado





Líneas de Productos Software

- Una aplicación (o familia de aplicaciones) se generaliza en una arquitectura común, que puede ser adaptada para diferentes clientes.
 - funcionalidades genéricas que pueden ser adaptadas y configuradas para su uso en un contexto específico
- La adaptación puede implicar:
 - Configuración del sistema
 - Selección en una biblioteca de componentes existentes
 - Adición/modificación de nuevos componentes para satisfacer las nuevas necesidades



Pasos en el desarrollo con Líneas de productos

- Elicitar los requisitos
- Encontrar el producto de la línea que mejor se adapte a los requisitos
- Renegociar los requisitos
- Adaptar el sistema existente
 - Desarrollar nuevos módulos y hacer cambios en la línea de productos
 - Documentar el desarrollo de las funciones del módulo adicional
- Entregar el nuevo producto de la línea

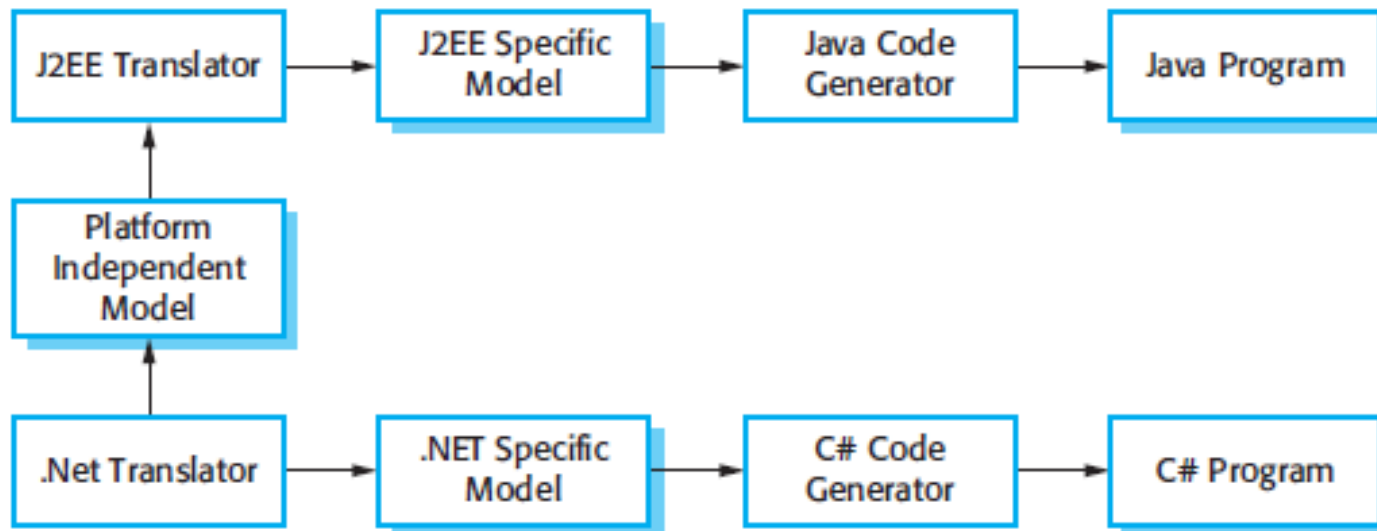


Lenguajes específicos de dominio

- Un lenguaje específico de dominio (domain-specific language, DSL) es un lenguaje dedicado a resolver un problema en particular (Textual o Gráficamente)
- Ejemplos
 - SQL para consultas a bases de datos
 - Hojas de cálculo
 - Mathematica
 - GUI-Builder
- Una Línea de productos se puede configurar con un lenguaje específico

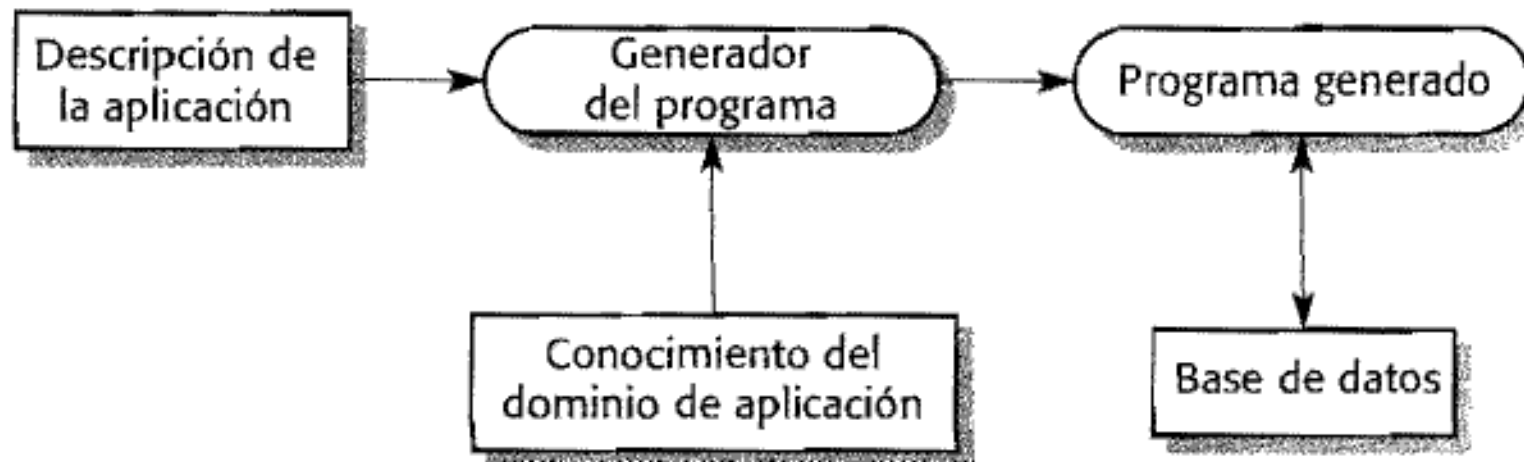
MDD/MDA

- Desarrollo Dirigido por Modelos (MDD)
 - Se desarrollan los modelos de dominio y el código se genera a partir de estos modelos.
 - Se basa en la Arquitectura Dirigida por Modelos (MDA), propuesta por el Object Management Group (OMG) en 2001 como un nuevo paradigma de desarrollo



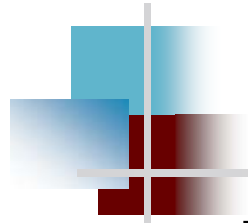
Generadores de programas

- El conocimiento reutilizable se captura en un sistema generador
- Programado por expertos en el dominio utilizando un lenguaje específico del dominio
- La descripción de la aplicación específica es un modelo abstracto



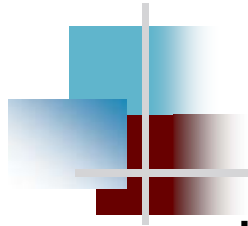
1.2. Beneficios de los Componentes





¿Componentes?

- Los componentes software representan una de las posibles soluciones de reutilización.
- Beneficios potenciales:
 - Cada componente adquirido es un producto **estandarizado**, con todas las ventajas, pero además en el proceso de ensamblado se ofrece la oportunidad de **adaptarlo**
 - Un producto que utiliza componentes combina la productividad e innovación de todos los vendedores de componentes



Composición y componentes

- La composición permite que cosas prefabricadas se usen en nuevas composiciones.
- Para convertir un elemento en reutilizable, no es suficiente tomar un diseño monolítico de una solución completa y dividirlo en fragmentos.
- Las descripciones de los componentes deben **ser generalizadas** cuidadosamente para permitir la reusabilidad en un número suficiente de diferentes contextos.



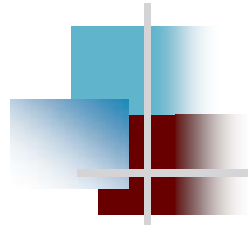
Diferencias con OO

- Los componentes software **son unidades ejecutables de: adquisición, implementación y despliegue independiente** que forman parte de ("componen") un sistema
- Los requisitos de independencia y ser ejecutables eliminan muchas construcciones software: declaraciones, macros de C, plantillas de C++...
- **Clases, módulos**, o incluso aplicaciones enteras pueden ser componentes... pero en una **forma ejecutable** que sea integrable (.DLL, .jar)



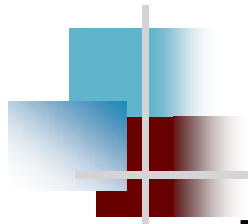
Evolución de la orientación a objetos

- La unidad de implementación es algo más complejo que los objetos.
 - OO –encapsulación de estado y el comportamiento, polimorfismo, y la herencia- no incluye **independencia o composición**
- Los Componentes añaden además:
 - Concepción de la **arquitectura** como un elemento de primer orden
 - La interacción de elementos **heterogéneos**.
 - Se destaca la importancia de las **interfaces**.
 - Mecanismos de comunicación \neq invocación tradicional



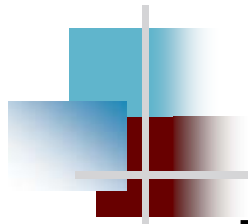
Además ... "se puede vender"

- Configurar e integrar un objeto individual no es posible
 - Los objetos no pueden ser vendidos independientemente como componentes
- Un componente debe tener un número considerable de usos y de clientes, para que sea comercialmente viable.
 - El "uso repetido" -> reusabilidad.



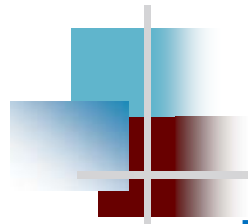
Ventajas

- Principal ventaja: **granularidad más gruesa**
 - Facilita la reutilización de implementaciones y diseños, a un nivel de granularidad mayor que la clase
- El diseño basado en **interfaces** facilita sistemas extensibles.
 - Se pueden añadir nuevos componentes, o añadir interfaces nuevas a los ya existentes.
- **Independencia de un lenguajes** de programación específicos
 - Usando componentes desarrollados en otros lenguajes → .NET
 - Interacción transparente entre componentes distribuidos.
 - Importancia de los estándares!



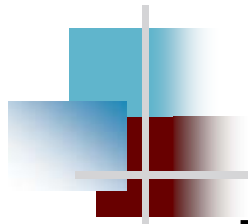
Ventajas prácticas

- Reutilización **binaria**
 - El producto final no necesita manipulación posterior (ej.: Compilación). Esto evita problemas de compilación
 - No alteración indeseada del código fuente
 - No revelación de algoritmos
- **Reducción de los costes** de desarrollo: sólo desarrollo de partes aún no desarrolladas por nadie (o caras)
- Más **velocidad** (menos tiempo) de construcción de nuevos sistemas.
 - Partes listas para ser ensambladas.



Ventajas prácticas

- Reducción del coste de mantenimiento y actualización de sistemas existentes,
 - Sustitución de partes defectuosas o antiguas por otras nuevas, de manera independiente.
 - Incluso, si se dispone de ligadura dinámica, se puede actualizar en tiempo de ejecución
- Facilita el desarrollo en paralelo de las distintas partes de un sistema, una vez establecido el conjunto de interfaces de cada componente
- Es más fácil aprovechar las últimas tecnologías y avances, al surgir proveedores especializados



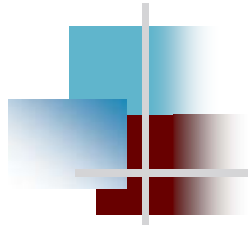
Riesgos

- No hay **control** sobre todas las partes del sistema
- **Peor rendimiento**
 - Niveles de indirección, adaptadores, interacciones remotas...
- **Integración** del sistema
 - Se hace más difícil (integración puede ser costosa. Ej.: un componente descargado de Internet).
 - La interacción entre elementos heterogéneos debe estar perfectamente prediseñada.
- Responsabilidades legales
 - Si un sistema falla, ¿cómo se determina el componente **culpable**?

1.2.1.

Evolución de la tecnología de componentes





Evolución

- Filtros y tuberías de Unix
- OLE y COM, Active X, JavaBeans
- JEE/EJB, COM+, .NET
- Servicios Web, SOA

Unix (filtros y tuberías)

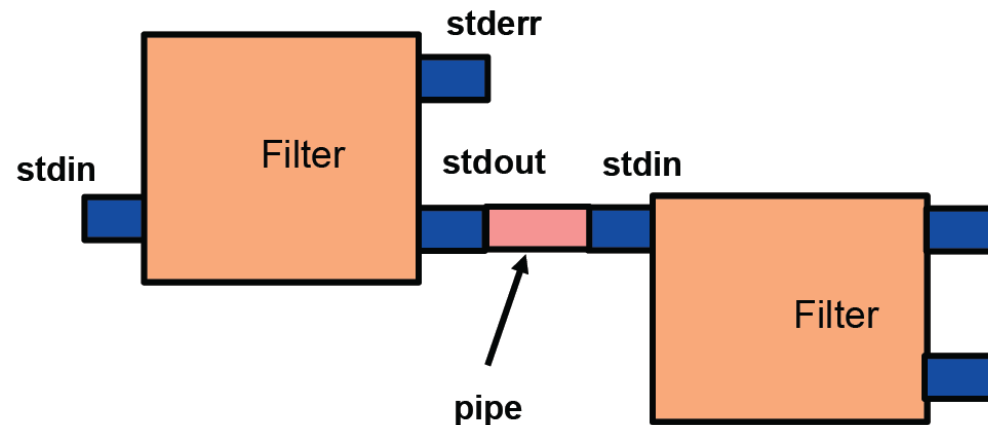
```
$ cat apple.txt
```

Ipad

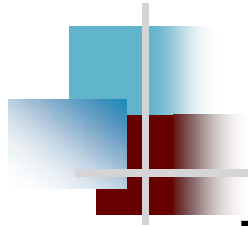
Iphone

Macbook

- Pero si usamos `|`



```
$ cat apple.txt | wc
```

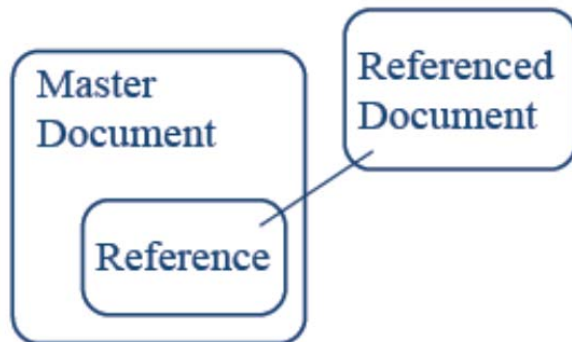


Unix (filtros y tuberías)

- Modelo de Componentes
 - Componentes desconocidos
 - Puntos de conexión: stdin/stdout
- Técnica de Composición
 - Lenguajes de filtros: sed, awk, grep, wc
- Lenguaje de Composición
 - Shell de Unix

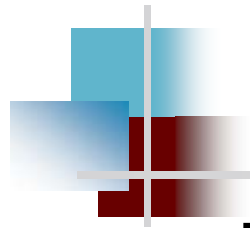
OLE, COM y ActiveX

- Tecnología Microsoft (componentes locales)
- Se puede insertar (o enlazar) una hoja de cálculo en un documento Word (OLE)
- O añadir controles (visuales o no) a un formulario o una página Web (ActiveX)



The screenshot shows a Windows form titled 'Form1' with a standard Windows XP-style title bar. Inside the form, there is a calendar control for July 2005. The calendar has a header with navigation arrows and the text 'July, 2005'. Below the header is a grid of dates. The date '20' is highlighted in blue. At the bottom of the calendar, there is a label 'Today: 7/20/2005' and a 'NEXT' button.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6



Componentes COM

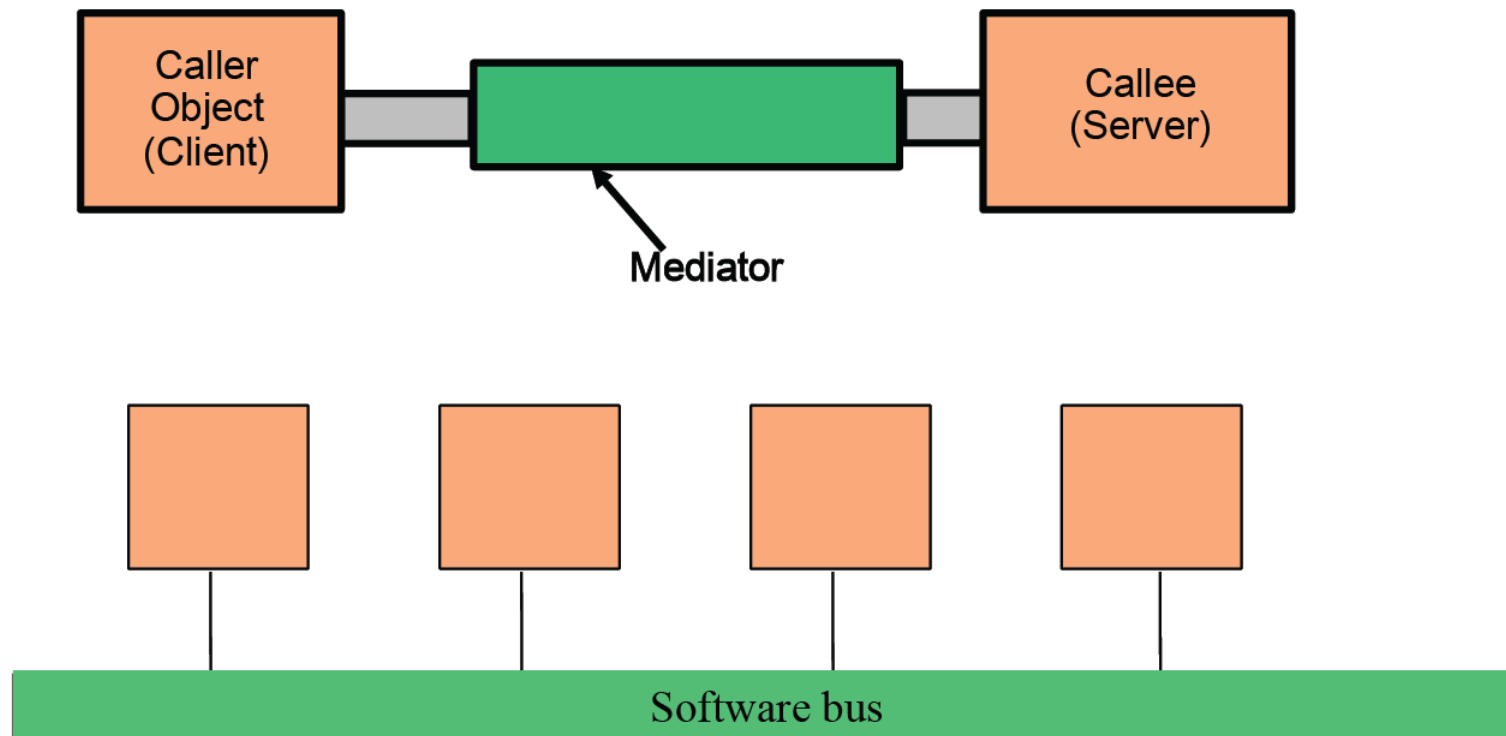
- Modelo de Componentes Local
 - Componentes binarios
 - Puntos de conexión están estandarizados
 - Descrito por un lenguaje de definición de interfaces o IDL
 - Interfaces estándar (IUnknown...)

- Técnica de Composición
 - Mezcla de lenguajes VB, C++

- Lenguaje de Composición
 - Visual Basic, VBA

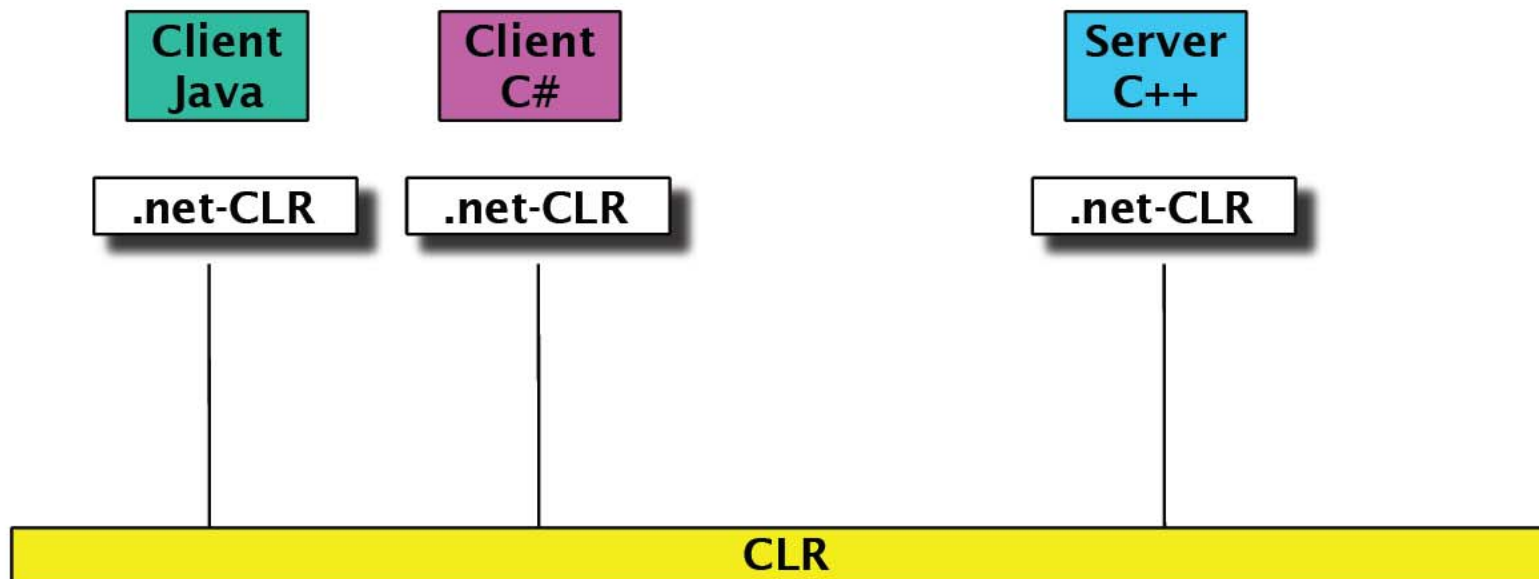
Componentes distribuidos: Corba, .NET y JEE/EJB

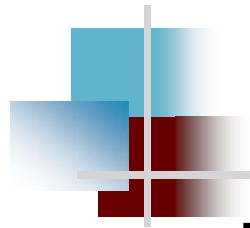
- Soporte middleware (puede estar integrado en el sistema operativo)



Ejemplo: .NET

- Independiente del lenguaje
- C# sirve como IDL
- CLR: lenguaje común en tiempo de ejecución



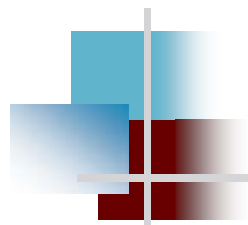


Componentes .NET y JEE/EJB

- Modelo de Componentes
 - Componentes binarios
 - Puntos de conexión están estandarizados
 - Descrito por IDL (C# o Java)

- Técnica de Composición
 - Sistemas distribuidos y [mezcla de lenguajes]

- Lenguaje de Composición
 - C#, Java...



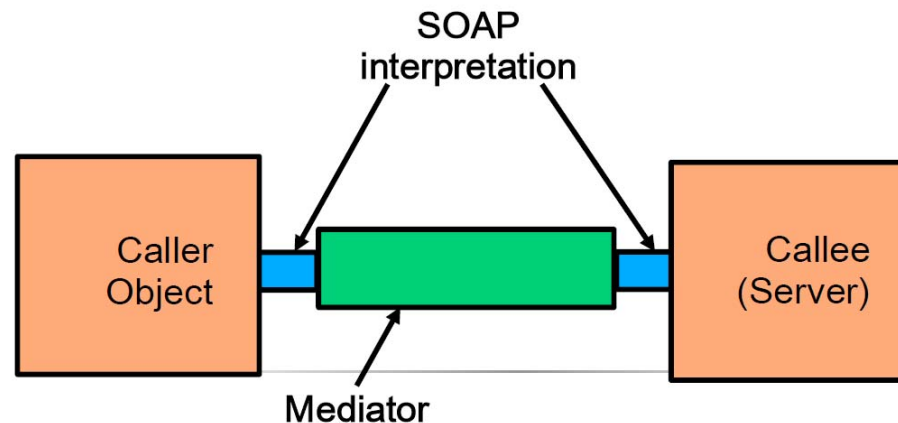
Servicios Web

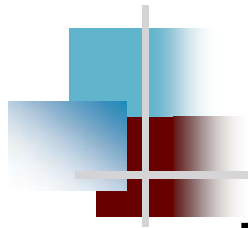
web2logo.com showing popular 50, 100, 150 logos 1 2 3 4 5 6 7 8 9 10 Next



Servicios Web

- Procedimiento de conexión **interpretado**, no compilado
- Más flexible:
 - Aunque cambie la interfaz, no hay que recompilar
 - **Protocolo HTTP** - independiente de .NET , CORBA, etc.





Servicios Web

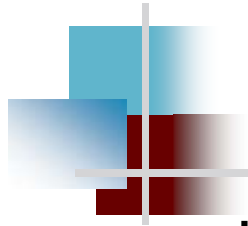
- Modelo de Componentes
 - Contenido: no es importante
 - Puntos de conexión: se describen con XML, JSON
 - Procedimiento de conexión es la interpretación de SOAP/WSDL o REST

- Técnica de Composición
 - Sistemas distribuidos y cualquier lenguaje
 - HTTP, WSDL/SOAP, JSON

- Lenguaje de Composición
 - Cualquiera
 - BPEL: Business Process Execution Language

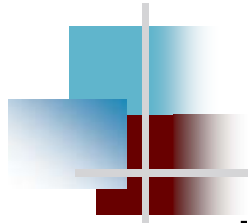
1.2.2. Mercado y estándares





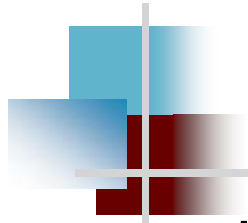
El mercado y la tecnología

- Los componentes son activos reutilizables.
 - Resolver un problema general en lugar de uno específico implica más trabajo.
 - Los componentes son viables sólo si la inversión en su desarrollo se recupera como un resultado de su venta/reutilización.
- Una tecnología imperfecta en un mercado que funciona es sostenible
- Una tecnología perfecta sin ningún mercado es inútil



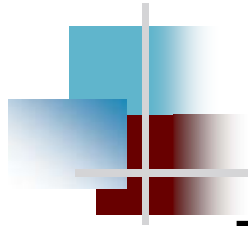
Crear un Mercado

- Un nuevo producto puede crear un mercado solo si su llegada ya se estaba esperando.
- La producción de componentes debe ser menos costosa que la producción de soluciones completas
- Además, el empaquetarlos con componentes relacionados disminuye los costes de distribución
 - Algunos componentes no son capaces de sostener solos los precios que los podrían hacer viables



Crear un Mercado

- Un camino seguro es el que evita crear mercados y se expande cuidadosamente en mercados ya establecidos (estrategia de Microsoft).
- Controles VBX
- Generalizados a controles OLE (OCX)
 - expandió el mercado de VB a todas las aplicaciones de escritorio.
- De controles OLE a controles ActiveX
 - aplicaciones de Internet.



Mercados de componentes

- Tres modelos comerciales
 - OMG: CORBA/CORBA Component Model
 - Oracle/Sun: JavaBeans, Enterprise JavaBeans (EJB)
 - Microsoft: COM/COM+ (OLE, ActiveX), .NET
 - <http://www.componentsource.com>

- Mercados de componentes y open source en Internet:
 - GNOME: Bonobo, KDE: Kparts...
 - Plug-ins para Eclipse y navegadores...

- Google Play (Android) es también un mercado de componentes (se conectan con los mensajes INTENT)

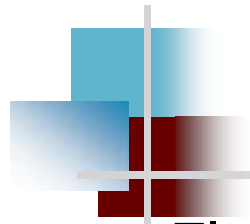
El desarrollo de un mercado

- **ComponentSource** sobre todo Microsoft pero también Java
- **Flashline** se enfoca en el desarrollo de componentes para el servidor y tiene preferencia por las tecnologías basadas en Java.

Productos ofrecidos en ComponentSource

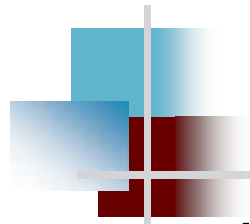
ComponentSource (Diciembre 30 2001)	
plataforma	category
COM	95
VCL	47
.NET	42
JavaBeans	40
EJB	12
CORBA	2
total:	

Products By Type (2014)	
• Components	(2389)
• .NET Components	(1549)
• ActiveX / COM	(432)
• HTML5 / jQuery / JavaScript / AJAX	(222)
• Flash / Flex	(11)
• C++ / MFC Class Libraries	(102)
• DLL	(327)
• Java Components	(211)
• VCL	(199)



Factibilidad tecnológica

- El establecimiento de un mercado de componentes depende de la tecnología
- En un mercado de desarrolladores independientes:
 - El conjunto de posibles combinaciones de componentes no es conocido por ninguna de las partes
 - Los componentes necesitan estar contruidos de tal manera que permitan una prueba modular
- ...Pero se debe garantizar:
 - La seguridad (si hay un fallo no se debe colapsar todo el Sistema)
 - La funcionalidad
 - El rendimiento



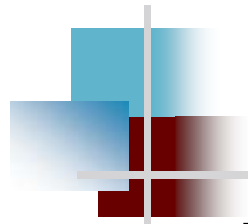
Componentes estándar

- Un componente necesita una porción significativa de un mercado específico para que sea rentable
- Para ello, necesita tener requisitos que sean comunes (y solicitados)
 - Si un componente sirve para pocos clientes, el vendedor conocerá exactamente las necesidades individuales (más fácil)
 - Cuando el número de usos y de clientes crece, es difícil que un componente pueda cubrir todas las necesidades
- El punto medio para clientes y vendedores es el que está basado en los estándares/acuerdos



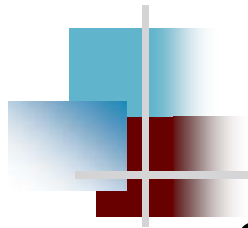
Estándares (de facto): casos de éxito

- Servicios verticales: modelos compartido por distintas organizaciones en un dominio concreto
 - Sterling Software (componentes genéricos de negocio banca y aseguradoras, conversión monetaria, gestión clientes, emisión informes financieros)
- Servicios horizontales: servicios comunes a varios dominios de negocio
 - Soporte al comercio electrónico en Internet: validación y verificación de la información que fluye por la red, acceso a plataformas electrónicas de pago, seguimiento en tiempo real de pedidos...



Estándares (oficiales)

- Los estándares establecen acuerdos entre distintos modelos de componentes, para su interoperabilidad
- Conjunto de normas estándar a las que adherirse (Ej. CORBA)
 - Equivalente a los componentes electrónicos: voltajes, medidas de clavijas: [interfaces](#)
 - Si no son compatibles necesitamos [adaptadores](#)
 - Se necesita un armazón ([framework](#)) para ensamblar los componentes
 - + Definición de servicios comunes-> [middleware](#)

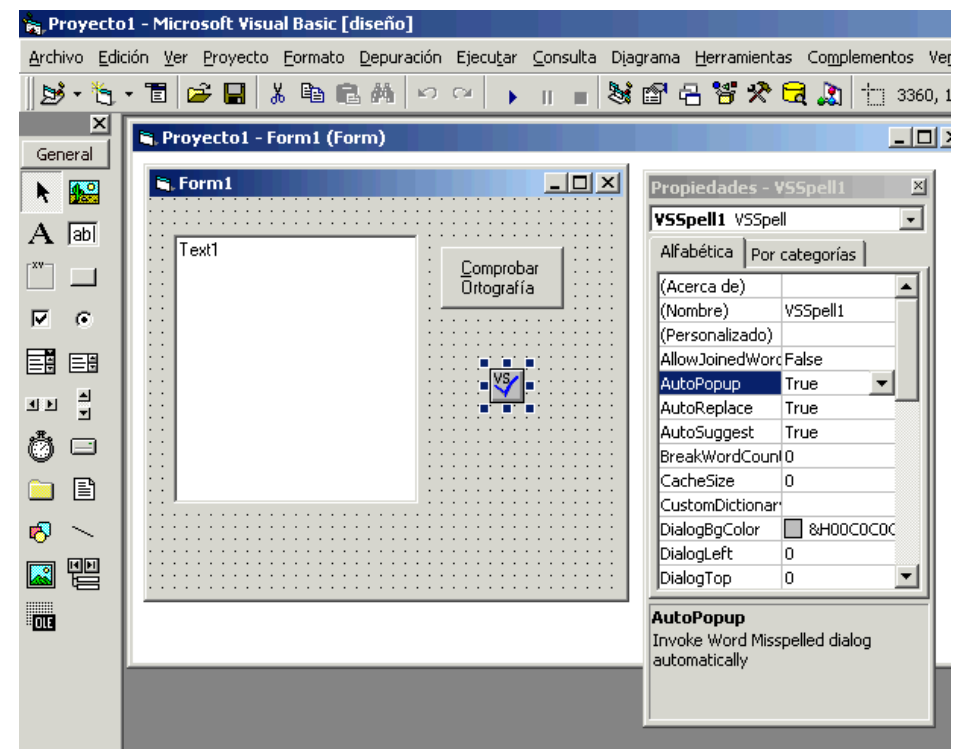


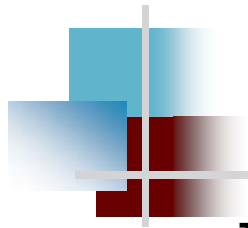
Estándares: middleware

- Servicios estándar de conexión:
 - mecanismos y protocolos de interacción: eventos, mensajes, reglas para asignar nombres a las operaciones, convenciones para utilizar servicios de otros componentes...
 - servicios de repositorio de interfaces y componentes (encontrar componentes o servicios disponibles) y referencia o directorio (nombres significativos para referirse a otros componentes de forma transparente)
 - servicios de entrega de mensajes (ej. entrega solo una vez)
 - gestión de transacciones que involucren a varios componentes
 - habilitación de comunicaciones seguras

Herramientas de soporte

- Conceptos de programación visual
 - Programación orientada a eventos
 - Entornos integrados (IDE): NetBeans, Visual Studio, Delphi
- Componentes visuales:
 - ventanas, diálogos, etiquetas,...





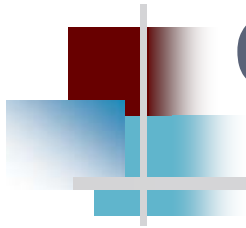
Herramientas

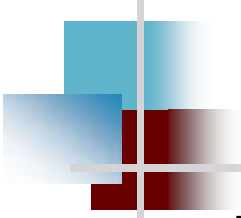
- Instalación de componentes
 - normalmente las paletas de componentes son extensibles

- Grupos de componentes importantes (familias)
 - acceso a bases de datos
 - creación de interfaces graficas
 - visualización de datos
 - ...

- Distribución e instalación
 - suelen incluirse asistentes para empaquetamiento (JAR, EAR...), registro automático (COM, .NET)
 - Herramientas de configuración (.NET)

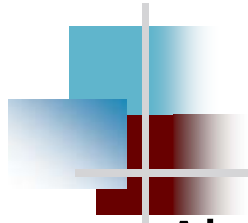
1.3. Fundamentos y definiciones





¿Qué es un componente y ...qué no lo es?

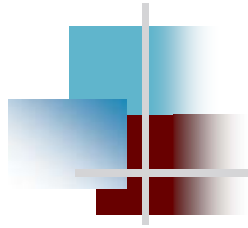
- Los términos **componente** y **objeto** a menudo se confunden...
- Sin embargo:
 - Un componente (binario, p. e. un control ActiveX) se puede implementar mediante lenguajes no orientados a objetos (C, VB)
 - Un componente puede encapsular un sistema existente (sistema legado) en una organización, que no tiene por qué ser OO



Componentes y objetos

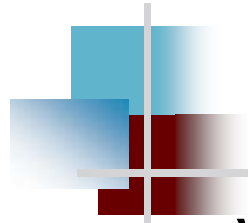
Ahora bien:

- Las similitudes hacen que la tecnología OO sea muy apropiada para implementar componentes.
- Un componente (plantilla) se puede diseñar e implementar mediante clases si se desarrolla con un lenguaje OO.
- Una vez instanciado, este componente ejecutable podrá estar compuesto internamente por varios objetos.



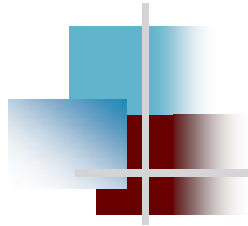
No hay una única opinión...

- Pfister y Szyperski
 - Ven un componente como una colección de objetos, en la que estos objetos cooperan unos con otros, y están entrelazados estrechamente
- D'Souza y Wills
 - Afirman que si una clase se empaqueta junto con las interfaces definidas explícitamente, entonces esta clase es un componente.



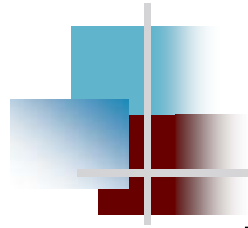
Componente: Definición

- “Unidad de composición de aplicaciones software, que posee un **conjunto de interfaces** y un conjunto de **requisitos**, y que tiene que poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma **independiente**, en tiempo y espacio.” (Szyperski)
- Especificaciones claras de lo que requiere y de lo que proporciona.
- Interacciona con su entorno a través de interfaces bien definidas.



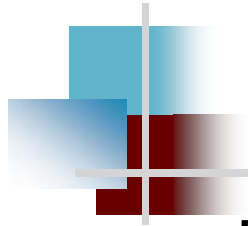
Definición Alternativa

- Porciones reutilizables de software, que se desarrollan independientemente, y que pueden combinarse con otros componentes para construir unidades más grandes. Puede ser adaptado, pero **no modificado**.
- Un componente puede ser, por ejemplo, código fuente compilado, pero también parte de un modelo o diseño.(D'Souza and Wills)



Propiedades características

- Es una unidad de implementación independiente.
 - Debe ser auto-contenido.
- Es una unidad que puede ser fabricada por terceras partes.
 - Los clientes no tienen porqué acceder a los detalles de construcción del componente.
- No puede ser parcialmente implementado



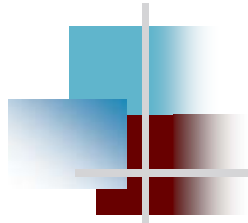
Propiedades características

- Dos elementos que lo describen:
 - Un conjunto de interfaces (las que **proporciona** y las que **requiere**).
 - **Código ejecutable** que se puede combinar con otros componentes a través de esas interfaces
- [No cuenta con un estado observable desde el exterior]
 - Excepto atributos no funcionales como el número de serie.
- [Son unidades “pesadas”, existe solo una copia]
 - Por tanto, en un proceso se puede decir si hay o no un componente, pero no varias instancias del mismo.



Objetos: recordatorio

- “Un objeto es una unidad de instanciación con una identidad única, un estado y un conjunto de operaciones. El estado esta representado por el conjunto de valores que toman las propiedades en un instante de tiempo, el cual varía dinámicamente como resultado de la ejecución de sus operaciones.”
- La clase es la plantilla genérica a partir de la cual se pueden instanciar los objetos
- En contraste con un componente, un objeto:
 - Es una unidad de instanciación, tiene una única identidad.
 - Puede tener estados y estos pueden ser observables externamente.
 - Encapsula su estado y comportamiento.



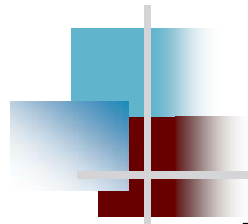
Por contra...los componentes

- A menudo utilizan **almacenamiento persistente** mientras que los objetos residen en memoria
- Tienen un conjunto más amplio de **mecanismos de intercomunicación** que los objetos (que utilizan el paso de mensajes)
- Suelen ser más grandes que los objetos, y tienen **acciones complejas** en sus interfaces



Ciclo de Vida de un Componente

- Especificación del Componente.
 - especificación de las interfaces que lo conforman y de manera independiente de la plataforma en la que va a ser construido
- Implementación del Componente.
 - se realiza la especificación en un entorno concreto, respetando los reglas que establece un modelo de componentes.
- **Despliegue** ("deployment") del componente.
 - disponibilidad del componente en la plataforma específica, para ser utilizado por las diferentes aplicaciones.
- Instanciación del Componente
 - en el momento de su utilización.



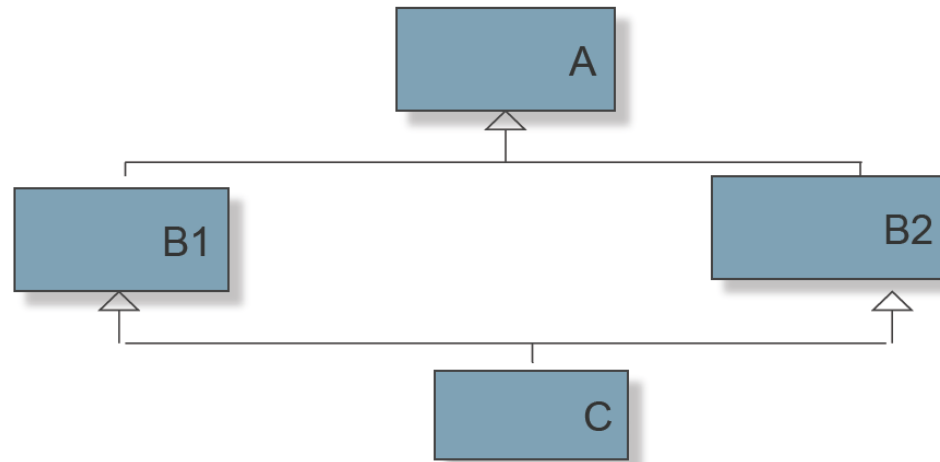
Implementación/Despliegue

- Un componente se puede implementar:
 - Clases, procedimientos tradicionales, y variables globales.
 - Puede ser realizado en su totalidad utilizando programación funcional

- Despliegue depende del contexto
 - Interfaces requeridas
 - Entorno de componentes para el cuál esta preparado (COM, CORBA, .NET, JEE)
 - Plataforma (Hardware/Software)

Herencia frente a composición

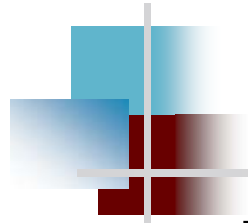
- Problemas con la herencia
 - Si hay herencia múltiple, se pueden heredar distintas implementaciones del mismo método





Problemas con la herencia

- ¿Puede una clase base evolucionar sin afectar a sus subclases que han evolucionado a su vez?
 - Esta estrecha dependencia de subclases desarrolladas independientemente de su clase base se conoce como el problema de la **clase base frágil**
- Puede ser sólo un problema sintáctico (hay que recompilar la subclase si cambia la clase base, no hay compatibilidad binaria) o un problema semántico (la clase base realmente ha cambiado en puntos clave)

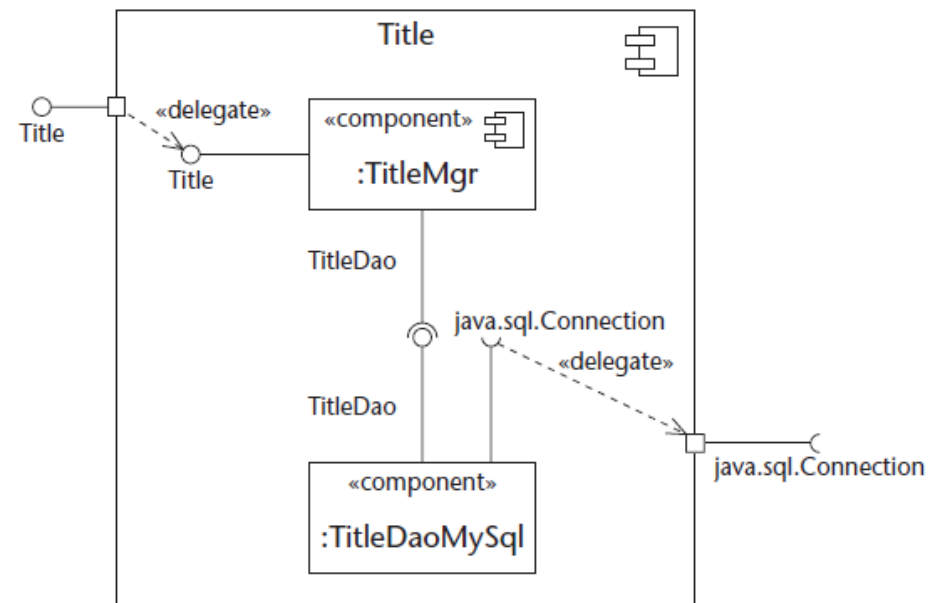
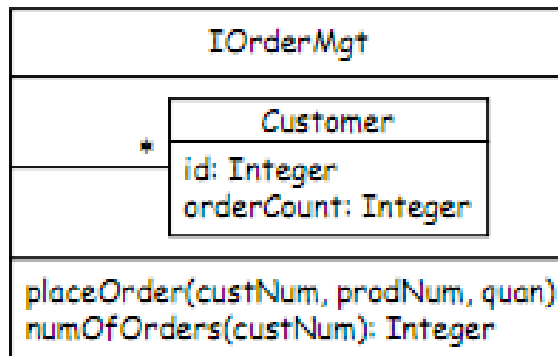


Herencia y componentes

- La herencia es un mecanismo de extensión de gran alcance
- La programación con herencia requiere métodos muy disciplinados.
- No es adecuado para la extensión independiente
- Si se usa, la herencia no debe cruzar las fronteras del componente

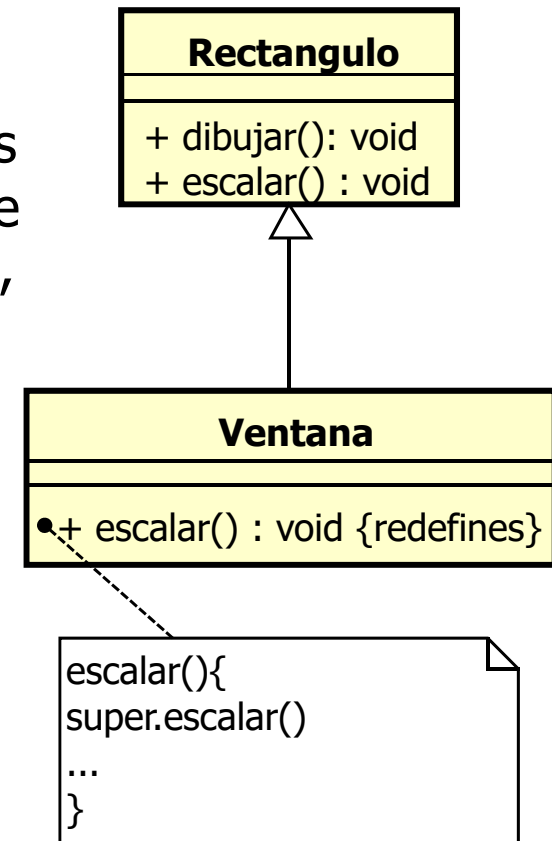
Composición

- Cuando un objeto necesita colaborar con otro, le envía una solicitud (mensaje)
- Si el objeto auxiliar es parte del mismo se habla de composición ("inner object")
- Es también una forma de reutilización



No hay "auto recursión implícita"

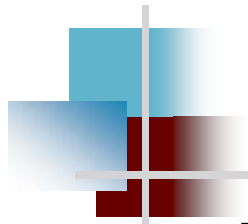
- Diferencia fundamental:
 - En la herencia, el objeto de la subclase es el mismo que la instancia de la superclase (ej. Si **Ventana** hereda de **Rectángulo**, la misma instancia es una **Ventana** y un **Rectángulo**).
 - Esto permite llamadas "auto-recursivas"
 - ej. **escalar()** de **Ventana** puede invocar a **escalar()** de **Rectángulo**
- Con la composición, los dos objetos son diferentes



1.4

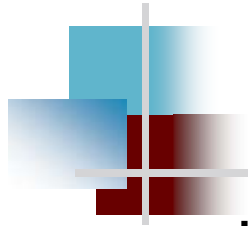
Interfaces y contratos

- Interfaces: variantes
- Especificación de contratos



Interfaces

- Determinan las **operaciones** que el componente **implementa** y las que **necesita utilizar** de otros componentes durante la ejecución.
 - (Incluye atributos y métodos públicos más los eventos que emite)
- Una interfaz es independiente de la implementación:
 - Permite cambiar la implementación sin cambiar la interfaz
 - Permite añadir nuevas interfaces (e implementaciones) sin cambiar la implementación existente



Especificación de la Interfaz

- La **especificación es un contrato** de su punto de acceso
 - El respeto de este contrato por parte del cliente y componente asegura el éxito de la interacción
- Para los usuarios debe consistir en:
 - Una definición precisa y completa de las operaciones del componente (única interfaz visible)
 - Todas las dependencias de contexto.
- Para los desarrolladores
 - La especificación de un componente proporciona una definición abstracta de su estructura/comportamiento internos



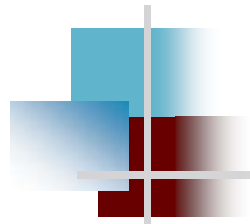
Dos tipos de interfaces

- Los componentes pueden exportar e importar interfaces a y desde el entorno, que incluirá a otros componentes.
- Una interfaz exportada describe los servicios **proporcionados** por un componente a su entorno
- Una interfaz importada especifica los servicios **requeridos** por el componente del entorno



Descripción (Sintaxis)

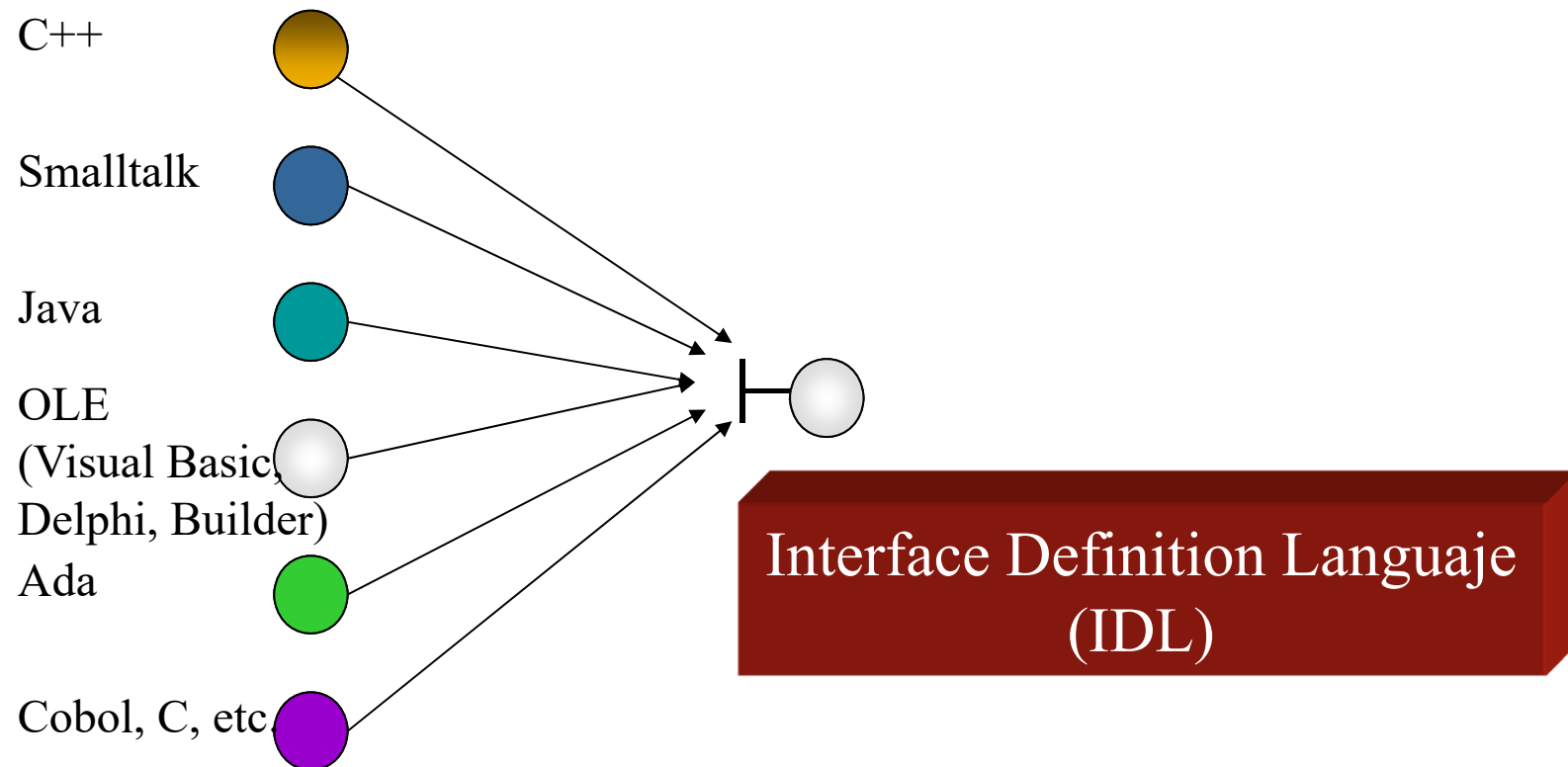
- Las interfaces se definen de forma estándar utilizando un lenguaje de definición de interfaces (IDL)*
 - Suficiente para la descripción de las propiedades funcionales.
 - Insuficiente para la descripción de las propiedades no funcionales como atributos de calidad como la precisión, disponibilidad, seguridad, etc.
- *incluyendo Java, C#.... o UML

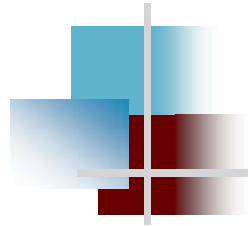


Estándares IDL

- Common Object Request Broker Architecture (CORBA)
- Microsoft's Component Object Model (COM, DCOM, COM+)
- UML
- .NET (C#)
- Sun JavaBeans y EJB (Java)
- WSDL (Servicios Web)

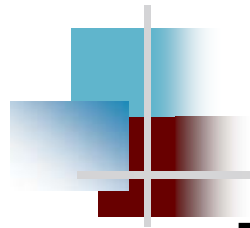
CORBA IDL





Características de Corba IDL

- OMG Common Object Request Broker Architecture
- Describe operaciones y parámetros de cada interfaz.
- Lenguaje declarativo.
- Sintáxis similar al ANSI C++.
- Preprocesado como C++ (más #pragma).
- Usa el código de caracteres ISO-Latin1.



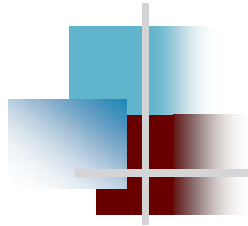
Tipos de datos estilo C++

- Tipos básicos:
 - Enteros y reales: short, long, float, double
 - Carácter: char, wchar
 - Booleano: boolean
- Otros:
 - Cadena: string, wstring
 - Secuencia: sequence
 - Estructura: struct
 - Unión: union
 - Enumeración: enum
- Vectores: tipo[n]



Operaciones

- Formato de una operación
 - [atributo] tipo identificador parámetros [excepciones]
- Atributos:
 - Sentido único: oneway
- Parámetros:
 - ([atributo tipo identificador [, ...]])
 - Atributos: in out inout
- Excepciones:
 - raises (excepción [, excepción])



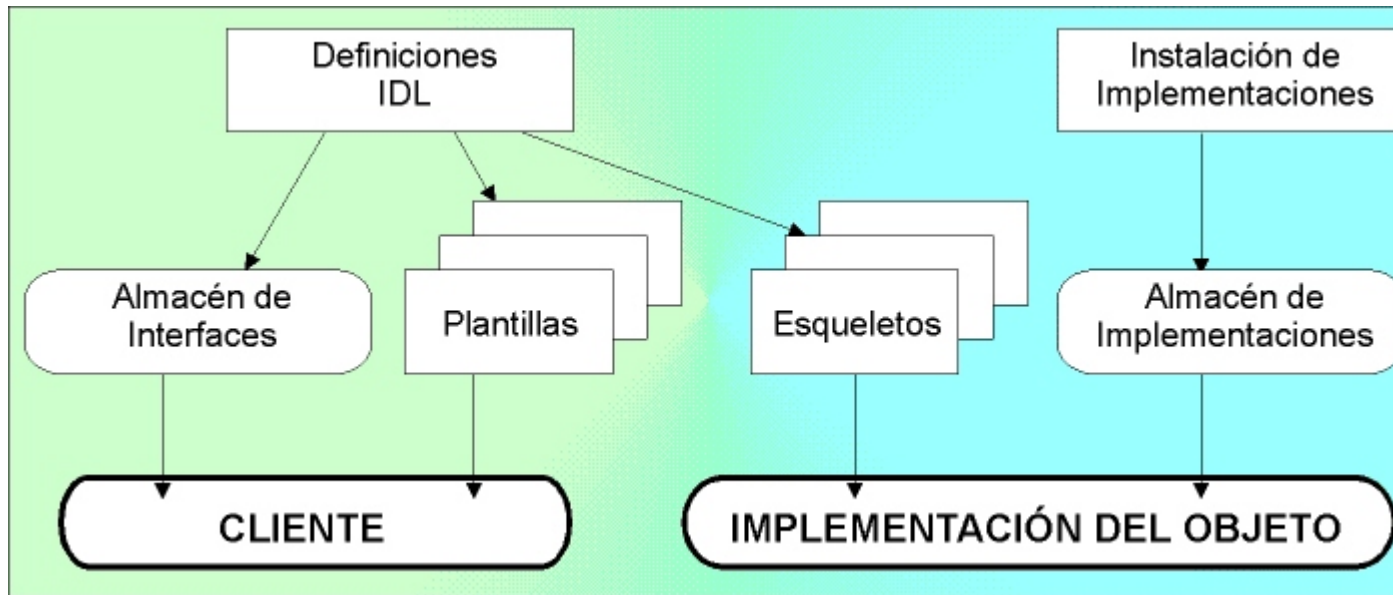
Ejemplo (fragmento)

Política de seguridad de un objeto:

```
module CORBA {  
    typedef unsigned long PolicyType;  
  
    interface Policy {  
        readonly attribute PolicyType policy_type;  
        Policy copy ();  
        void destroy ();  
    }  
    typedef sequence<Policy> PolicyList;  
}
```

CORBA: Interfaces

- Las definiciones IDL de las interfaces se usan para generar interfaces y esqueletos del código en lenguajes convencionales (C++)



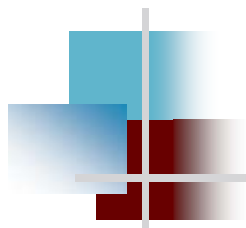


MS IDL (Microsoft COM)

```
interface ISpellCheck : IUnknown
{
    HRESULT check([in] BSTR *word, [out] bool *correct);
};

interface ICustomSpellCheck : IUnknown
{
    HRESULT add([in] BSTR *word);
    HRESULT remove([in] BSTR *word);
};

library SpellCheckerLib
{
    coclass SpellChecker
    {
        [default] interface ISpellCheck;
        interface ICustomSpellCheck;
    };
};
```

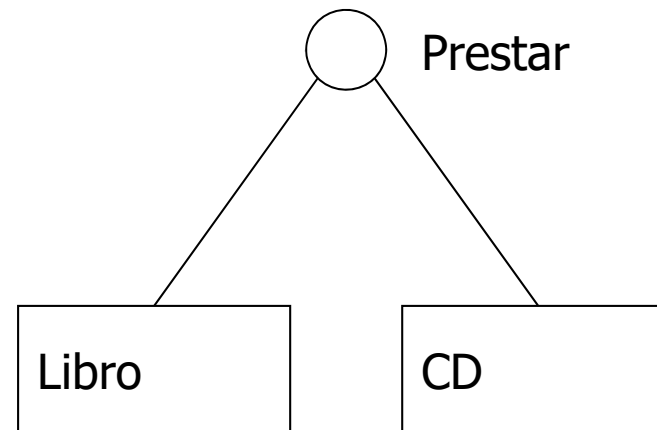
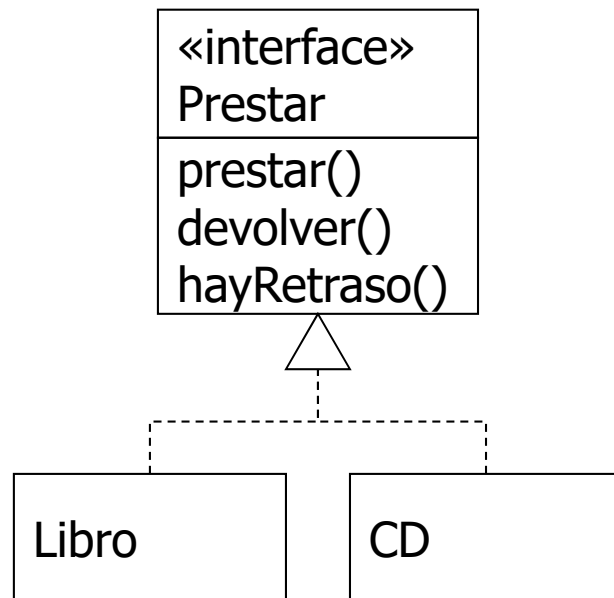
IDL (COM)

B i t	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1</
-------------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

- HRESULT es un código de error
- S - Severity - success/fail
 - 0 - Success
 - 1 - Failure
- C - Customer.
 - 0 - Microsoft-defined
 - 1 - Customer-defined
- Facility - Servicio
 - 1 - RPC
 - 2 - Dispatch
 - 3 - Storage
 - 4 - ITF
 - 7 - Win32
 - 8 - Windows

UML: interfaz proporcionada

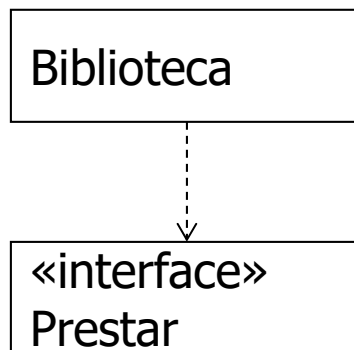
- Una clase o componente implementa una interfaz y por tanto proporciona sus servicios



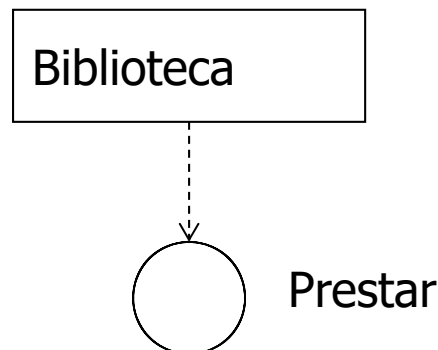
UML: Interfaz requerida

- Una interfaz requerida indica que un clasificador utiliza los servicios definidos por la interfaz
- Tres formas de expresar la dependencia de <<uso>>

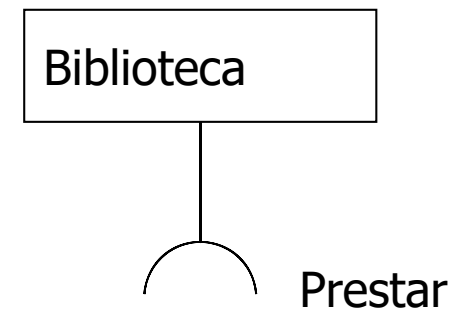
estándar básico



UML 1.x

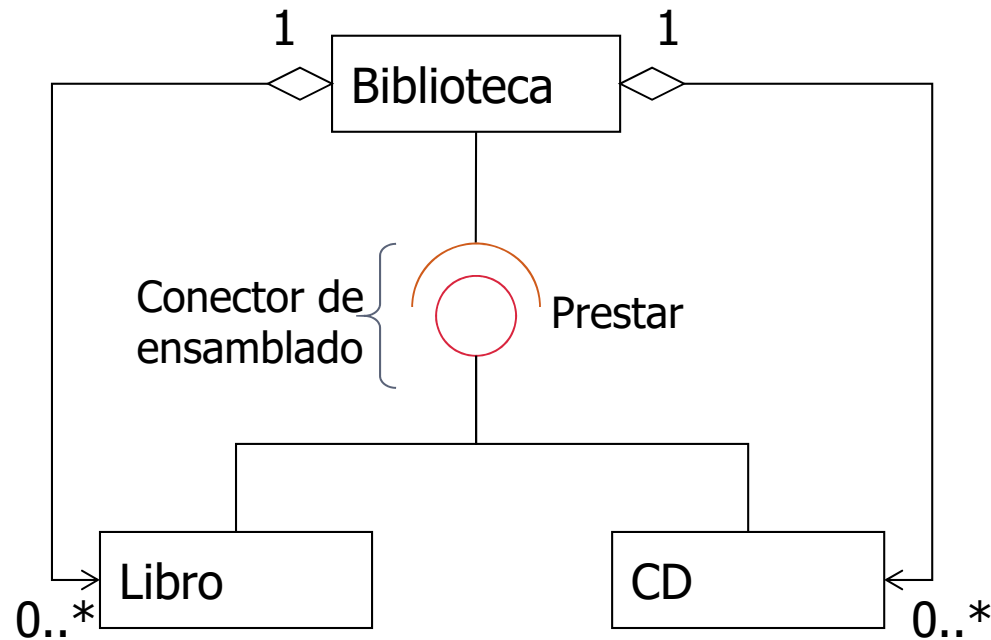


UML 2.x

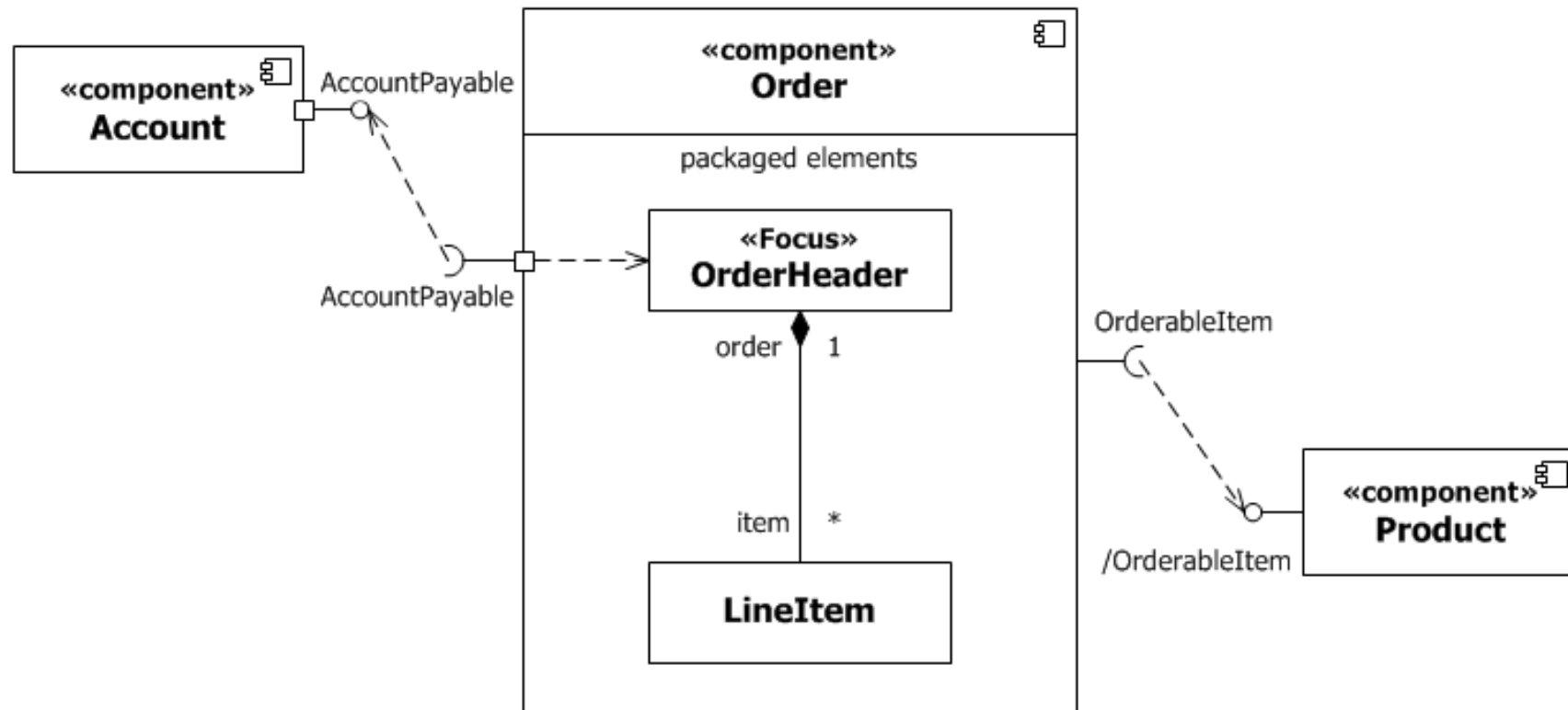


Conectores de ensamblado

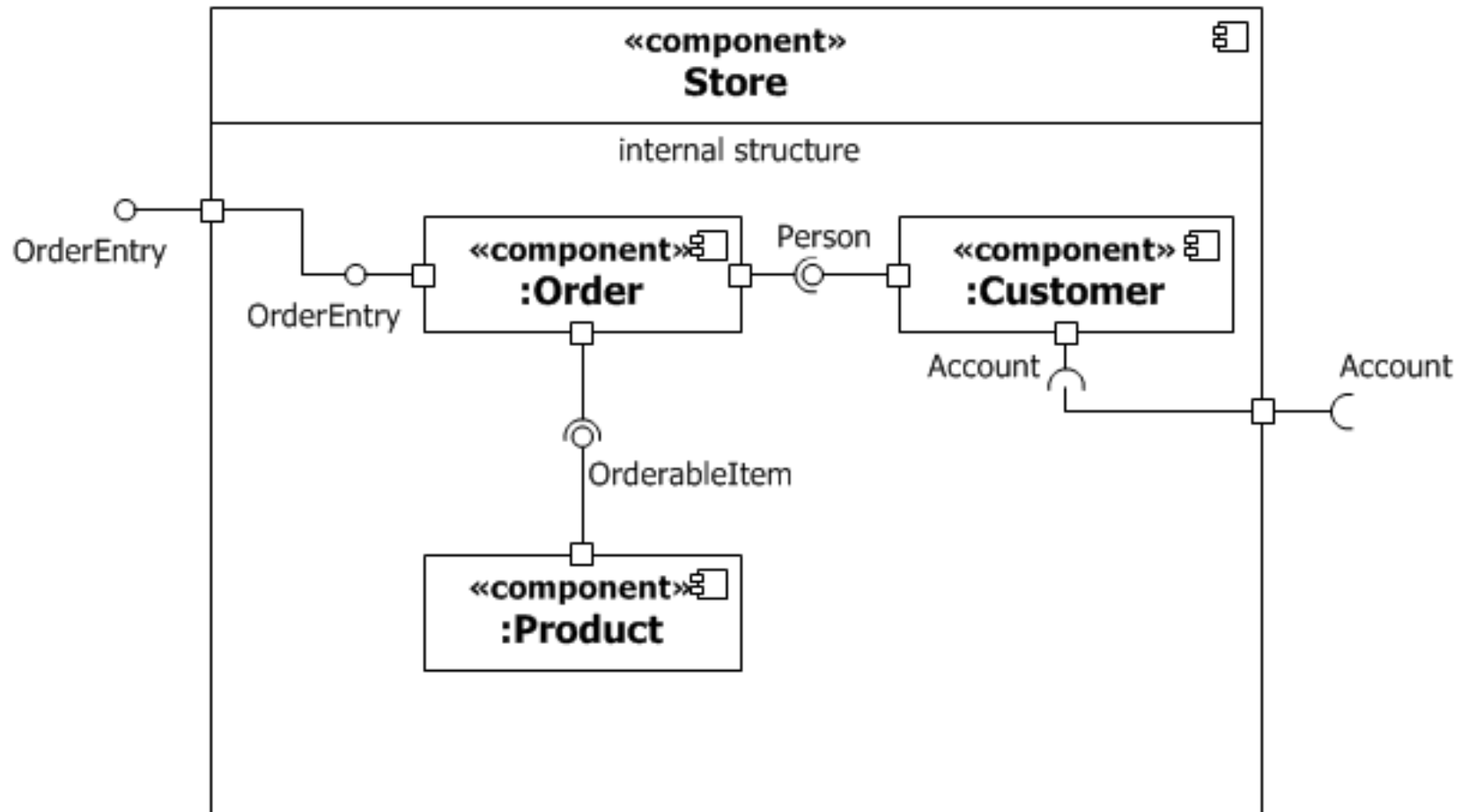
- Se pueden conectar ambos tipos de interfaz

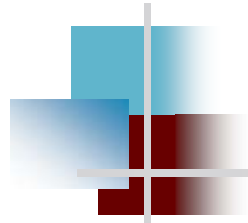


Internamente habrá clases...



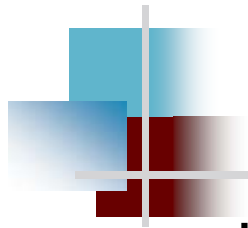
...o incluso otros componentes





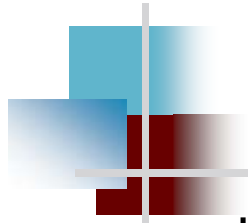
Utilidad de las interfaces

- Comprobación de tipos en el código de cliente.
- Una base para la interoperabilidad entre componentes desarrollados y otras aplicaciones.
- Un aspecto importante de las especificaciones de la interfaz es cómo se relacionan con la sustitución y la evolución de los componentes...



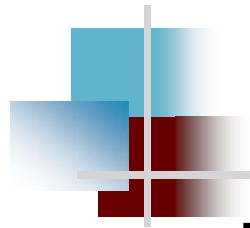
Substitución

- La substitución de un componente Y por un componente X se dice que es segura si:
 - Todos los sistemas que funcionan con X también trabajarán con Y
- Desde un punto de vista **sintáctico**, un componente puede ser reemplazado [con seguridad] si:
 - El nuevo componente implementa al menos las mismas interfaces que el componentes antiguo
 - La interfaz del nuevo componente es un subtipo de la interfaz del componente antiguo



Semántica: Contratos

- La especificación más precisa del comportamiento de un componente se puede lograr mediante invariantes, pre y post-condiciones
- Invariante:
 - Es una expresión sobre el estado de la interfaz que siempre se cumple
 - Se puede asociar un conjunto de invariantes con una interfaz.



Pre y post-condición

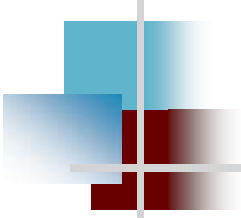
■ Pre-condición

- Es una afirmación que el componente asume que debe cumplirse antes de invocar la operación
- En general, es una expresión con los parámetros de entrada de la operación [y su estado]

■ Post-condición

- Es una afirmación de lo que el componente garantiza que se cumplirá después de que la operación haya sido invocada (siempre que la pre-condición fueran cierta)
- Es una expresión con los parámetros de entrada y salida, [así como el estado inmediatamente antes y después de la invocación]

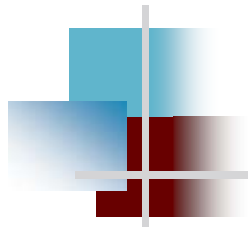
Especificación de contratos: OCL



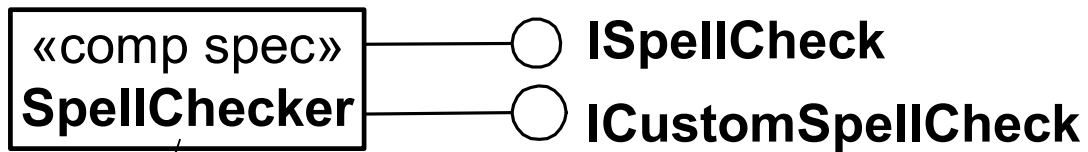
```
interface ICustomSpellCheck : IUnknown
{
    HRESULT add([in] BSTR *word);
    HRESULT remove([in] BSTR *word);
};
```

```
context ICustomSpellCheck::add(in word : String) : HRESULT
pre: word <> ""
post: SUCCEEDED(result) implies
        words = words@pre->including (word)
```

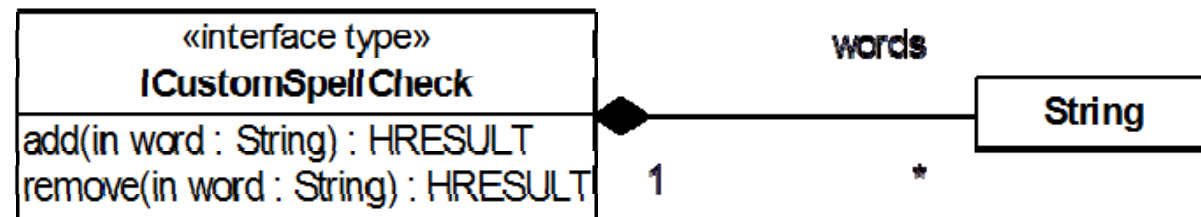
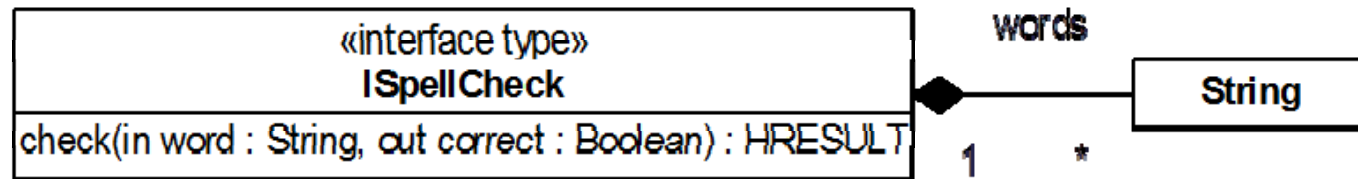
```
context ICustomSpellCheck::remove(in word : String) : HRESULT
pre: word <> ""
post: SUCCEEDED(result) implies
        words = words@pre->exluding(word)
```



Diagramas UML + OCL



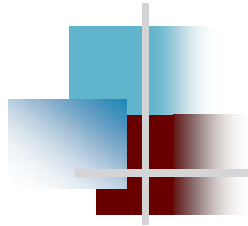
{inv: ISpellCheck::words = ICustomSpellCheck::words}



context `ISpellCheck::check(in word : String, out correct : Boolean) : HRESULT`
pre: `word <> ""`
post: `SUCCEEDED(result) implies correct = words->includes(word)`

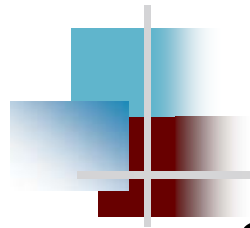
context `ICustomSpellCheck::add(in word : String) : HRESULT`
pre: `word <> ""`
post: `SUCCEEDED(result) implies`
`words = words@pre->including (word)`

context `ICustomSpellCheck::remove(in word : String) : HRESULT`
pre: `word <> ""`
post: `SUCCEEDED(result) implies`
`words = words@pre->exluding(word)`



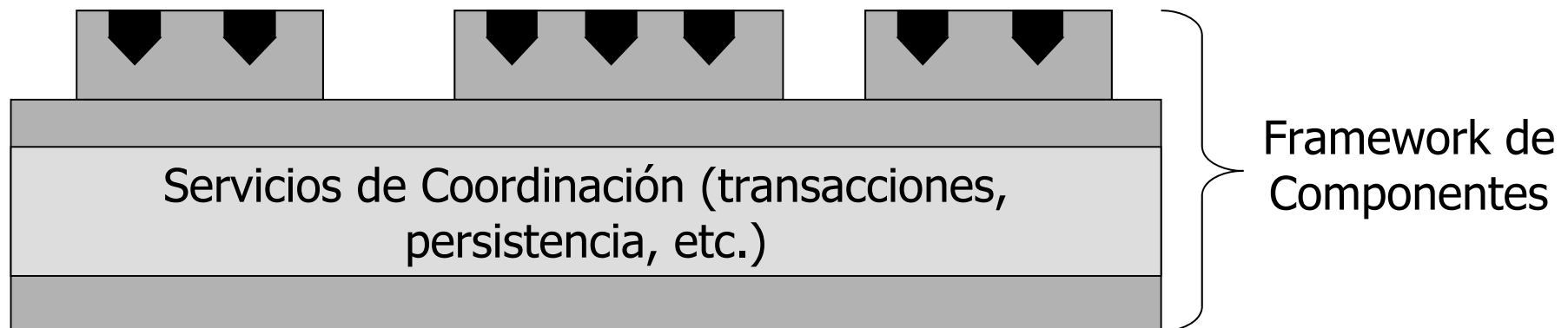
El papel de los Frameworks

- Un Framework, en general, es:
 - Un diseño reutilizable de un sistema,
 - Un esqueleto de una aplicación que puede ser personalizada por el desarrollador de aplicaciones
- La idea de componentes es que construimos software “juntando piezas”
- Un Framework proporciona el armazón o infraestructura en el que las piezas se pueden insertar



Frameworks de Componentes

- Si los frameworks en general describen un diseño reutilizable, un Framework de componentes es **también una "placa base"** con ranuras vacías en el que los componentes se pueden insertar para crear un sistema que funcione





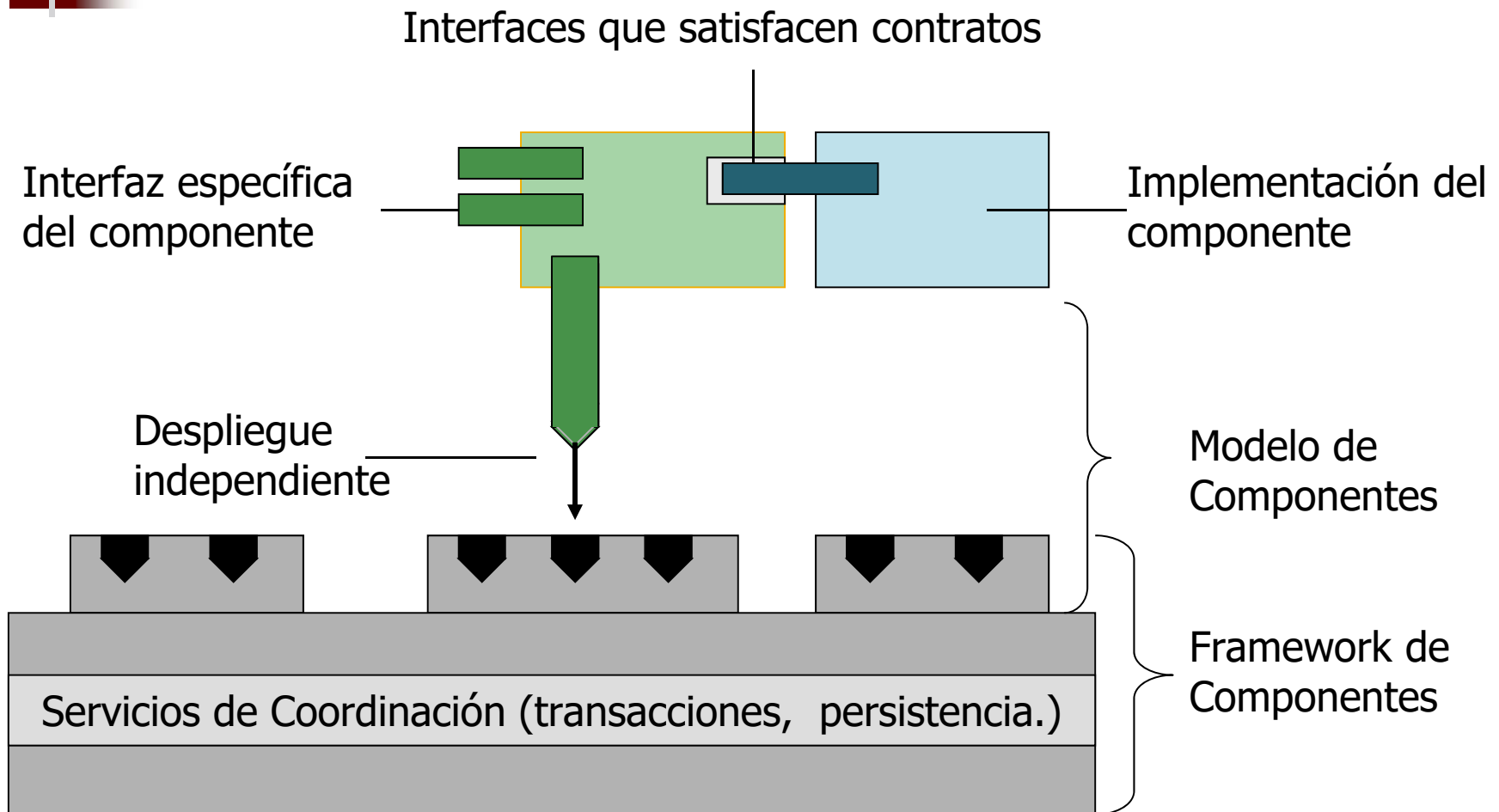
Modelos de Componentes

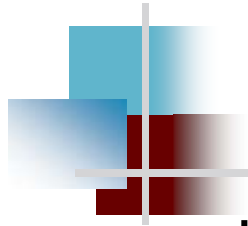
- Modelos de Componentes y Frameworks de Componentes a veces se entremezclan
 - Un Modelo de Componentes define un conjunto de estándares y convenciones usadas por los desarrolladores
 - Un Framework es una infraestructura de soporte para ese modelo



Modelo de
Componentes

Relaciones entre conceptos





Frameworks y Contratos

- Los frameworks también imponen obligaciones, pero..
 - Los frameworks se centran en las propiedades globales de las composiciones
 - Los contratos son especificaciones para las relaciones concretas entre los componentes.

1.5.

Ingeniería de Software Basada en Componentes (CBSE)



(Sommerville)



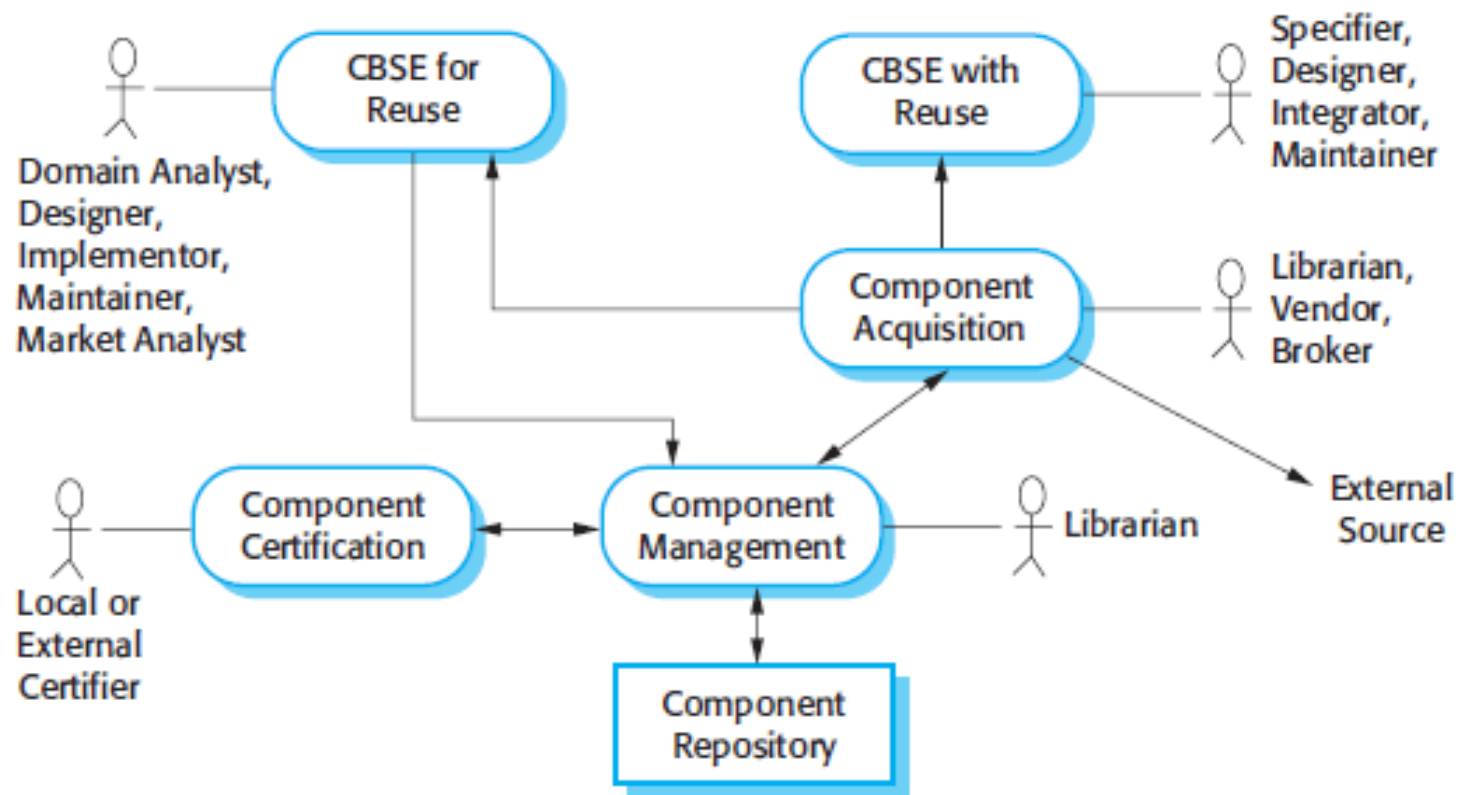
Ingeniería de Software Basada en Componentes

- En CBSE (Component Based Software Engineering) el desarrollo es un trabajo de adaptación y composición a partir de componentes
- Estos componentes pueden tener diversos orígenes: desarrollados para uso genérico, comprados o desarrollados a la medida

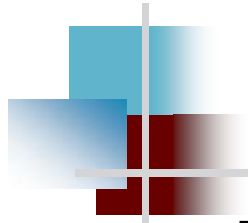
■ Objetivos

- Desarrollar sistemas a partir de componentes ya contruidos.
- Desarrollar componentes como entidades reutilizables.
- Mantener y evolucionar el sistema a partir de la adaptación y reemplazo de sus componentes.

Procesos CBSE

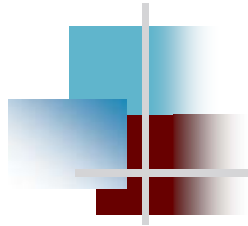


(Sommerville 9^a ed.)



Procesos CBSE

- Incluyen las diferentes actividades implicadas en el desarrollo y el uso de componentes reutilizables:
- Desarrollo para reutilización
 - El desarrollo de componentes o servicios que se pueden reutilizar en otras aplicaciones.
 - Por lo general, implica la generalización de los componentes existentes.
- Desarrollo con reutilización
 - Desarrollo de nuevas aplicaciones utilizando componentes y servicios existentes



Procesos de soporte

- Adquisición de componentes
 - adquirir componentes para su reutilización
 - pueden ser componentes desarrollados localmente o provenientes de una fuente externa.
- Gestión de componentes
 - asegurar su correcta catalogación, almacenamiento y puesta a disposición para su reutilización.
- Certificación de componentes
 - verificar y certificar que un componente cumple con su especificación.