



*FEED YOUR BRAIN®*

# Arquitectura SOA con tecnología Microsoft

**Buenas prácticas y diseño  
de aplicaciones empresariales.**

César de la Torre  
Roberto González

Prólogo de Jose Murillo, Responsable de Programas Técnicos para Partners, Microsoft Ibérica

**Arquitectura SOA con  
Tecnología Microsoft**

---

César de la Torre  
Roberto González



**César de la Torre:**

“A mi familia, Marta, Erika y Adrián, por aguantar todos los días que trabajo en casa! ;-)”

**Roberto González:**

“A mi mujer, por llevar tan bien el nacimiento de nuestro primer hijo a la vez que el de este libro y por supuesto a José Manuel Alarcón y a César de la Torre por darme la oportunidad de participar en este proyecto”.



**ARQUITECTURA SOA CON TECNOLOGÍA MICROSOFT**  
**Buenas prácticas y diseño de aplicaciones empresariales**

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito de los titulares del Copyright. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, [www.cedro.org](http://www.cedro.org)) si necesita fotocopiar o escanear algún fragmento de esta obra.

DERECHOS RESERVADOS © 2008, respecto a la primera edición en español, por

Krasis Consulting, S. L.

[www.krasis.com](http://www.krasis.com)

ISBN: 978-84-935489-7-1

Depósito Legal: C -2008

Impreso en España-Printed in Spain

# Prólogo

Sin duda un libro de cabecera para todos aquellos Arquitectos de Software que tengan ante sí la excitante tarea de construir una aplicación empresarial fiable, escalable y segura. Cada vez son más numerosas las aplicaciones empresariales que utilizamos en nuestro día a día, no solo en nuestro trabajo sino en nuestro tiempo libre, sobre todo Aplicaciones Web Empresariales, de reserva de viajes, compras online, y un largo etcétera. Al margen de cuestiones de usabilidad y diseño, todos hemos padecido en algún momento las aplicaciones poco fiables donde nuestra transacción se pierde irremediablemente a medio camino tras varios minutos introduciendo datos, o sistemas no escalables donde los tiempos de espera nos hacen pulsar desesperadamente sobre la tecla F5 (Actualizar) del explorador. La diferencia entre una buena Aplicación Empresarial y las últimas mencionadas la marca a menudo la arquitectura, y sobre esas cuestiones se profundiza en este libro.

El primer capítulo comienza con una introducción a la Arquitectura Orientada a Servicios (SOA), repasando los primeros paradigmas de programación y sus diferentes evoluciones hasta los modelos de software distribuidos más habituales. Al final del primer capítulo se introduce al lector en conceptos como el Business Process Management (BPM) y el Enterprise Service Bus, al que se dedica un capítulo completo al final del libro.

Los dos siguientes capítulos llevan de la mano al lector en la programación de servicios Webs, desde conceptos básicos como SOAP y WSDL hasta modelos mucho más avanzados como la programación de Servicios Web asíncronos o la implementación de las especificaciones WS\* mediante la extensiones de las cabeceras SOAP. Todo con un primer enfoque teórico pero perfectamente complementado a continuación con ejemplos prácticos sobre servidores y tecnologías Microsoft.

Los tres capítulos siguientes sumergen al lector por una apasionante viaje a través de Windows Communication Foundation (WCF), el nuevo framework de Microsoft para la construcción de Servicios Webs y Arquitecturas SOA. De nuevo comienza por los conceptos más básicos, siempre didácticamente reforzados con ejemplos prácticos, hasta adentrarse rápidamente en los recovecos de esta tecnología con cuestiones mucho más avanzadas y potentes como la depuración y la seguridad de estos servicios. El último de estos tres capítulos se centra en las novedades de WCF con .NET 3.5, como las mejoras en el Entorno de Desarrollo, el soporte para REST, AJAX y RSS/ATOM, y el magnífico binomio Windows Communication Foundation y Workflow Foundation.

Los últimos capítulos que cierran el libro son sin lugar a duda mis preferidos. El primero de ellos aborda las buenas prácticas (y las que debemos evitar) de las arquitecturas SOA, algo que difícilmente encontraréis en otros libros de referencia de Web Services o SOA. Temas apasionantes como el versionado de servicios, los patrones y anti-patrones, cuestiones fundamentales de seguridad y la gestión de excepciones. El segundo nos introduce en el WCF Lob Adapter SDK, un componente que nos ayudará tremendamente en la exposición de aplicaciones de negocio (LOB) a través de Web Services. El último dedica páginas en exclusiva al Enterprise Service Bus, y nos instruye en la creación de arquitecturas SOA mediante la unión de Servicios Webs a través de este bus común.

Y para no robaros más tiempo antes de la lectura de este libro, una breve reseña en el plano más personal de los autores. Tengo el placer de conocerlos hace años y de haber trabajado junto a ellos en múltiples ocasiones, con Cesar de la Torre tengo la gran suerte de haberlo hecho muy estrechamente en los últimos meses y espero que continúe siendo así por muchos años. Cuentan con una dilatada experiencia en la arquitectura de aplicaciones empresariales y un gran dominio de productos Microsoft para la implementación de las mismas. A ellos solo mandarles un fuerte abrazo y toda la suerte del mundo en su trayectoria profesional y sobre todo personal. A vosotros desearos que aprendáis mucho con el libro, pero sobre todo que os divirtáis aún más con nuestra magnífica profesión.

*Jose Murillo*

Responsable de Programas Técnicos para Partners

Microsoft Spain

# Contenido

<b>I. Desarrollo Distribuido y SOA.....</b>	<b>I</b>
Evolución del desarrollo de software.....	2
Paradigmas de Programación.....	2
Programación estructurada .....	2
Programación modular.....	3
Programación orientada a objetos.....	4
Software distribuido.....	6
Cliente-Servidor .....	6
Arquitectura en tres Niveles (N-Tier).....	7
Ventajas de los sistemas distribuidos .....	8
Desventajas de los sistemas distribuidos .....	8
¿Qué es SOA? .....	9
¿Qué es un Servicio? .....	10
Los pilares de la Orientación a Servicios.....	11
Las fronteras de los Servicios deben ser explícitas .....	11
Los Servicios deben ser Autónomos .....	12
Los Servicios deben compartir Esquemas y Contratos, no Clases y Tipos ..	13
La Compatibilidad de los servicios se debe basar en Políticas .....	14
SOA vs OOP (Orientación a Servicios vs. Desarrollo orientado a Objetos distribuido tradicional) .....	15
Arquitectura interna de los Servicios SOA .....	16
Conceptos clave de servicios web.....	17
WSDL.....	18
SOAP.....	18
Mensaje SOAP.....	19
UDDI .....	21
Introducción a la gestión de procesos de negocio (BPM).....	22
¿Qué es BPM?.....	22
SOA + BPM .....	22
Introducción a integración de aplicaciones empresariales (EAI).....	23
Integración centralizada .....	23
Enterprise Services Bus.....	25

x Contenido

<b>2. Servicios Web XML básicos .....</b>	<b>27</b>
Implementación de Servicios Web XML básicos ASMX 2.0 con Visual Studio 2005/2008.....	31
Desarrollo de Servicio Web básico con VS.2005/2008: 'Hola mundo' .....	31
Consumo de Servicios Web XML básicos desde una aplicación cliente.....	35
Añadir una referencia web a nuestra aplicación .....	36
Consumo del Servicio-Web con las clases proxy .....	37
¿Qué son las clases proxy? .....	38
<b>3. Programación avanzada de Servicios Web .....</b>	<b>41</b>
Consumo de Servicios Web basado en Agentes.....	41
Invocaciones asíncronas de servicios web ASMX 2.0 .....	43
Cabeceras SOAP.....	44
Extensiones SOAP.....	46
Especificaciones WS* .....	49
Introducción a Servicios Web Avanzados con WSE (Web Services Enhancements).....	52
Historia y cronología de WSE.....	53
<b>4. Introducción a WCF (Windows Communication Foundation) ....</b>	<b>55</b>
¿Qué es WCF? .....	55
El 'ABC' de Windows Communication Foundation.....	58
Definición e implementación de un servicio WCF .....	60
1.- Definición del Contrato de un Servicio WCF.....	61
2.- Implementación del Servicio WCF .....	62
3.- Definición de contratos de datos (Data Contracts).....	63
Hospedaje del servicio (hosting) y configuración (Bindings) .....	65
Configuración del servicio.....	67
Desarrollo de Clientes WCF.....	75
Configuración de endpoints de cliente .....	75
Generación de clases Proxy cliente.....	76
Generación de clase proxy y configuración con SvcUtil.exe.....	76
Generación de clase proxy y configuración con Visual Studio .....	78
Tipos de hosting y como implementarlos .....	80
Hosting en Aplicación de Consola .....	80
Hosting en Servicio Windows.....	81
Hosting en IIS (Internet Information Server) .....	82

Hosting en IIS 7.0 - WAS.....	83
Hosting en Aplicación de Formularios Windows ó WPF.....	84
<b>5. Conceptos avanzados de WCF.....</b>	<b>85</b>
Tipos de Bindings en WCF.....	85
Binding P2P y aplicaciones Peer-To-Peer.....	87
Debugging y Trazas de mensajes en WCF.....	88
Debugging en WCF.....	88
Trazas de mensajes en WCF.....	88
Seguridad en WCF.....	90
Autenticación con WCF.....	93
Acceso anónimo.....	93
Autenticación con Token de seguridad Windows/Kerberos.....	94
Token de seguridad Usuario/password.....	97
Autenticación con token de seguridad Membership (tipo especial usuario/password).....	98
Autenticación con token de seguridad Certificado X.509.....	103
Autenticación con token de seguridad IssuedToken (CardSpace, SAML, etc.).....	103
Autorización con WCF.....	105
Autorización con Grupos de Windows/AD.....	107
Autorización con Roles de 'ASP.NET Role providers'.....	108
Autorización con Certificados cliente X.509.....	110
Conclusiones de Seguridad en WCF.....	112
<b>6. Novedades de WCF en .NET 3.5 (Visual Studio 2008).....</b>	<b>113</b>
Mejoras en el IDE de VS.2008 hacia WCF.....	113
Nuevos <b>bindings</b> en WCF 3.5.....	114
Soporte REST en WCF.....	116
¿Qué es REST?.....	117
La URI en REST.....	118
Simplicidad.....	118
URLs lógicas versus URLs físicas.....	119
Características base de Servicios Web REST.....	119
Principios de Diseño de Servicios Web REST.....	120
WCF y REST.....	121
Conclusión y recomendación sobre REST.....	121
Servicios WCF compatibles con AJAX.....	122

**xii** Contenido

Creación de un servicio WCF "AJAX <b>enabled</b> " .....	123
Configuración de Servicio WCF "AJAX <b>enabled</b> " .....	124
Soporte RSS y API de sindicación ATOM.....	125
Escenarios .....	126
Modelo de objetos.....	126
Creación de un SyndicationFeed.....	127
"Workflow Services" - Servicios WCF implementados con Workflows WF... 129	
Qué se necesita para integrar WCF y WF.....	130
Servicios WCF Duraderos (Durable Services).....	133
WCF en .NET Compact Framework 3.5.....	136
<b>7. Windows Communication Foundation Lob Adapter SDK.....</b>	<b>139</b>
Introducción.....	139
¿Qué es WCF Lob Adapter SDK?.....	139
Componentes de WCF LOB adapter SDK .....	140
Arquitectura de WCF LOB Adapter SDK .....	142
¿Qué método usar, servicios, canales o adaptadores?.....	143
Programando un adaptador mediante WCF LOB Adapter SDK .....	145
Instalación del sdk.....	145
Configuración del asistente de creación del adaptador paso a paso.....	146
Clases generadas por el asistente.....	152
Clase principal del adaptador.....	152
Implementación del canal .....	152
1. Creación del binding .....	152
2. Enlace del binding con el sistema de configuración de WCF.....	154
3. Configuración de propiedades del adaptador .....	155
4. Exposición del adaptador como un elemento del binding .....	157
Gestión de la conexión con el sistema destino.....	157
1. Gestión de la conexión.....	157
2. Definición y creación de la conexión.....	160
3. Propiedades de conexión con el sistema destino .....	163
Gestión de metadatos .....	165
1. Navegación jerárquica por los metadatos del sistema destino .....	165
2. Búsqueda de metadatos del sistema destino.....	169
3. Resolución de metadatos del sistema destino.....	171
Envío de mensajes al sistema destino .....	175
Recepción de mensajes del sistema destino .....	179
Despliegue del adaptador .....	182

Compilación del adaptador .....	182
Registro del adaptador.....	183
Consumo del adaptador.....	184
Consumo de operaciones de tipo Outbound.....	186
Consumo de operaciones de tipo Inbound.....	188
<b>8. Buenas prácticas en el desarrollo orientado a servicios .....</b>	<b>191</b>
¿Cómo modelar servicios?.....	191
Aísla la lógica de negocio del propio servicio .....	191
Mensajes vs. lista de parámetros .....	192
Versionado de contrato de datos.....	194
Versionado de servicios .....	195
Validación de parámetros .....	196
Gestión de excepciones.....	197
Patrones de intercambio de mensajes.....	199
Request-Response .....	199
One-way.....	201
Duplex.....	201
Seguridad.....	202
Autenticación.....	202
Autorización.....	204
Control de recursos y mejora del rendimiento.....	207
Patrones de desarrollo de servicios.....	209
Patrones de seguridad.....	209
Intranet.....	209
Internet.....	214
Anti-patrones.....	220
Anti-patrón: ¿Qué es lo nuevo?.....	221
Anti-patrón: Interface de creación, lectura, modificación y borrado (CRUD).....	222
Anti-patrón: Chatty interface.....	223
Anti-patrón: Super servicio.....	224
Anti-patrón: Integración punto a punto.....	225
Recomendaciones Top-10 de desarrollo de Servicios Web y Servicios SOA ...	227
<b>9. ESB E ISB .....</b>	<b>229</b>
Introducción.....	229
Contenido de la guía de Microsoft ESB.....	231

**xiv** Contenido

---

Servicios Web de ESB .....	231
Portal de Gestión de ESB .....	232
Componentes de interoperabilidad .....	233
Framework de gestión de excepciones.....	233
ESB Resolver and Adapter Provider Framework.....	233
Procesamiento de Itinerario de ESB.....	234
Arquitectura de la guía de Microsoft ESB .....	235
Ciclo de vida de un mensaje.....	235
Internet Service Bus (ISB).....	236
Software as a Service (SaaS) y Software plus Services (S+S).....	236
ISB y S+S.....	237
Implementación ISB de Microsoft: Biztalk Services .....	238
La filosofía de diseño de Biztalk Services .....	239
Detalles internos de Biztalk Services .....	240

# Desarrollo Distribuido y SOA

## ¿POR QUÉ ESTE LIBRO?

Este libro surgió como idea para ofrecer un soporte inicial a desarrolladores .NET que aunque hayan trabajado programando diferentes tipos de aplicaciones (como aplicaciones web ASP.NET, aplicaciones de formularios Windows, acceso a datos, etc.) sin embargo no han llegado a desarrollar aplicaciones **SOA**, es decir, aplicaciones con una **arquitectura orientada a servicios**, bien simplemente porque no han tenido la oportunidad o incluso porque lo ven como un mito indefinido y no se tiene claro como implementarlo.

En Internet existe un número incontable de información al respecto, e incluso hay bastantes libros teóricos sobre SOA y otros muchos específicos sobre tecnologías (.NET, WCF, etc.), pero la gran mayoría están en inglés y casi ninguno engloba todo ello de una forma práctica, es decir, cómo implementar aplicaciones SOA con tecnología Microsoft. Es realmente difícil localizar un libro en español que sea introductorio y global sobre SOA y al mismo tiempo sea un libro práctico que enseñe con ejemplos .NET como desarrollar Servicios Web tanto básicos como avanzados y además proponga recomendaciones de diseño, patrones y mejores prácticas. En definitiva, queremos ofrecer una ayuda útil y sencilla para quien quiere empezar.

Para ese perfil es precisamente para quien está orientado este libro, para alguien experimentado en programación .NET (C#, VB.NET, etc.) pero que sin embargo SOA y los servicios Web sean todavía un mundo por explorar o por lo menos algo de lo que se quiera comprender mejor todo su alcance.

La estructura del libro es por lo tanto muy progresiva e indicada para dicho perfil. Básicamente, veremos primero una introducción global a todo ello, es decir una “vista de pájaro” de SOA y el desarrollo distribuido. Una vez ahí, empezaremos con lo más sencillo, cómo implementar y consumir servicios web básicos .NET (Servicios Web **ASMX**). Posteriormente investigaremos algunos aspectos más avanzados de los servicios web básicos y a continuación seguiremos con la parte más importante a nivel de tecnología Microsoft para implementar SOA, es decir, **WCF (Windows Communication Foundation)**, desde sus conceptos básicos hasta aspectos avanzados,

seguridad, e incluso un capítulo específico de **WCF LOB SDK**. Un poco antes de finalizar el libro, volvemos a subir a un nivel global con un capítulo orientado a mejores prácticas, recomendaciones y patrones de desarrollo en arquitecturas orientadas a servicios. Realmente es otra “vista de pájaro”, pero en esta ocasión es necesario tener claro los conceptos SOA y de tecnología para poder aprovechar y entender realmente estas recomendaciones finales que hemos aprendido a lo largo de años y que así mismo están la mayoría definidas y contrastadas por la comunidad de desarrolladores. Estas recomendaciones finales y patrones son muy útiles a la hora de tener que implementar un proyecto real.

Finalmente y como colofón, hacemos una breve introducción al concepto de **ESB** (*Enterprise Service Bus*), muy importante en grandes implantaciones SOA en empresas con cierto volumen y donde se quiere tener una composición de servicios homogénea y bien estructurada.

## EVOLUCIÓN DEL DESARROLLO DE SOFTWARE

Antes de entrar directamente en SOA y en cómo podemos implementarlo, nos parece interesante enmarcar la situación actual dentro de la evolución que el desarrollo de software ha sufrido desde sus inicios hasta la actualidad.

Esta introducción es útil para aquel desarrollador que lleve pocos años programando y quizás no tenga una visión clara del típico tópico: “De dónde venimos y a donde vamos” en el mundo del desarrollo de software.

Por el contrario, si llevas muchos años en el mundo de la programación, por supuesto, puedes saltarte estas primeras páginas, aunque te puedes divertir recordando “viejas verdades” y paradigmas.

### Paradigmas de Programación

Los paradigmas de programación son enfoques particulares para el desarrollo del software. Son distintas maneras de visualizar y resolver problemas de programación.

En este libro no nos decantaremos por un paradigma o por otro, debido a que todos son igual de válidos y cada uno tendrá sus pros y sus contras. Dependiendo del caso en el que nos encontremos el sentido común y la experiencia nos dirá cual es el paradigma que se adapta mejor a la situación.

### **Programación estructurada**

En la década de los 60 surgieron los principios de lo que sería la programación estructurada. Es una forma de desarrollo en la cual solo está permitido el uso de tres lógicas de control:

- **Secuencia:** bloque de sentencias que se ejecutan una a continuación de otra.
- **Condicional:** bloque de sentencias que se ejecutan solo si se cumple una determina condición.
- **Interacción:** repetición de una o varias sentencias mientras se cumpla una condición dada.

En la programación estructurada se prohíbe el uso de las tan temidas sentencias GOTO (sentencias de salto incondicional) que tanto dificultaban la comprensión del código.

Un programa se puede decir que es estructurado si cumple las siguientes condiciones, entre otras:

- Posee un único punto de entrada y un único punto de salida
- Todas y cada una de las sentencias del programa son ejecutables, es decir no hay código muerto que no se ejecute en ninguna casuística.
- No hay bucles infinitos.
- Todos los posibles caminos llevan desde el punto de entrada al de salida.

Los programas desarrollados con este paradigma eran mucho más fáciles de entender que los desarrollados mediante una programación desestructurada y por lo tanto se aumentaba exponencialmente la productividad a la hora de mantener las aplicaciones.

Ahora bien, si seguimos este paradigma de programación nos encontramos con que vamos a tener un único módulo o bloque de código completamente inmanejable cuando el programa es demasiado grande, eso es lo que posteriormente denominamos “Aplicación Monolítica”. Es decir, es una programación estructurada, pero está implementado todo en un único programa monolítico, consiguiendo una reutilización de lógica y componentes, casi nula, además de ofrecer una localización y resolución de problemas muy costoso y por lo tanto un mantenimiento y evolución de la aplicación que dejan mucho que desear.

La forma de resolver este problema es mediante la programación modular que veremos en el siguiente punto.

### **Programación modular**

Cuando leemos algún libro de metodología de programación, una de las primeras técnicas que nos recomiendan es el algoritmo de **divide y vencerás** o programación **top-down**. Un problema se divide en partes más pequeñas que pueden ser abordadas de una forma más sencilla.

Usamos subprogramas estructurados que llamaremos **módulos** (de ahí el nombre del paradigma de programación) que interactúan entre sí para resolver el problema planteado.

#### 4 Arquitectura SOA con Tecnología Microsoft

---

Tendremos un módulo principal que coordine las llamadas a otros módulos secundarios (mediante el uso de procedimientos y funciones). La comunicación entre estos módulos se realiza mediante el intercambio de parámetros.

De esta forma tendremos un módulo que recibe en la llamada una serie de parámetros y que tras su procesamiento devuelve un resultado en la salida la cual puede ser tomada como entrada de otro módulo.

Cada módulo tiene la ventaja de que es reutilizable y puede ser considerado una caja negra (abstracción) de esta forma conseguimos independencia entre los módulos.

Un ejemplo de lenguaje de programación modular es el **lenguaje C**.

#### **Programación orientada a objetos**

La programación orientada a objetos se popularizó en la década de los 90 y es el paradigma de programación más utilizado actualmente.

Este paradigma permite resolver problemas mediante el trabajo colaborativo de objetos. Se intenta modelar objetos del mundo real en nuestras aplicaciones dando lugar al concepto de objeto.

Un objeto tiene propiedades y comportamiento:

- **Propiedades:** cada uno de los datos (atributos) que tiene el objeto.
- **Comportamiento:** cada una de las operaciones (métodos) mediante las cuales podemos interactuar con el objeto.

Una clase es el conjunto de propiedades y comportamientos de un objeto específico.

Podemos decir que una clase es la estructura en la que nos vamos a basar para crear el objeto. En la clase se representan los atributos y operaciones necesarias:

```
class Empleado
{
    string DNI;
    int numEmpleado;
    string NombreEmpleado;
    void AltaEmpleado(string DNI, int numEmpleado,
string NombreEmpleado)
    {...}
    ...
}
```

#### **Ejemplo de clase**

Mientras que el objeto es la representación en memoria de una instancia en concreto de la clase.

Por ejemplo:

```
Empleado objEmp = new Empleado();
objEmp.DNI="xxxxxxxxxxxx";
```

#### Ejemplo de objeto

Las principales características de la programación orientada a objetos son las siguientes:

- **Abstracción:** la abstracción se basa en la obtención de las características esenciales de un objeto. Por ejemplo que características comunes tiene el objeto *Empleado*.
- **Encapsulamiento:** es la unión en una clase de las características y comportamientos. Vemos a las clases como cajas negras donde solamente se hace público que es lo que hace pero no como lo hace. El acceso a esta caja negra es controlado mediante el ocultamiento (cada objeto expone un interfaz donde indica cómo se puede interactuar con él).
- **Herencia:** una clase no es una entidad aislada sino que pueden relacionarse entre sí formando una jerarquía. Retomando el ejemplo del *Empleado*, podemos tener una clase “padre” *Empleado* de la cual herede la clase *Directivo*. De esta forma estamos creando una clase especializada en base a una preexistente.
- **Polimorfismo:** cuando hablamos de polimorfismo en la programación orientada a objetos podemos referirnos a dos cosas:
  - Posibilidad de almacenar objetos de un determinado tipo en variables de tipos antecesores del primero.

Ej. Tenemos una clase *Círculo* que hereda de *Figura*, por lo tanto podemos asignar el contenido del objeto *Círculo* en un objeto *Figura* y usar cualquiera de sus métodos:

```
Figura fig = new Figura();
Figura fig2 = new Círculo();
fig.Dibujar(); //Dibujará una figura
fig2.Dibujar(); //Dibujará un Círculo
```

- Posibilidad de tener diferentes métodos dentro de una clase con el mismo nombre pero con diferentes argumentos.

## 6 Arquitectura SOA con Tecnología Microsoft

---

```
double sumar (int op1, int op2) {...}
double sumar (double op1, double op2) {...}
```

### ¿Cuáles son las ventajas de un lenguaje orientado a objetos?

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Agiliza el desarrollo de software.
- Facilita el trabajo en equipo.
- Facilita el mantenimiento del software.

### Software distribuido

El siguiente paso lógico en la programación de aplicaciones es el desarrollo distribuido. Podemos definir el software distribuido como un sistema cuyos componentes están ubicados en diferentes máquinas (normalmente servidores) y que se comunican entre sí mediante la transmisión de mensajes.

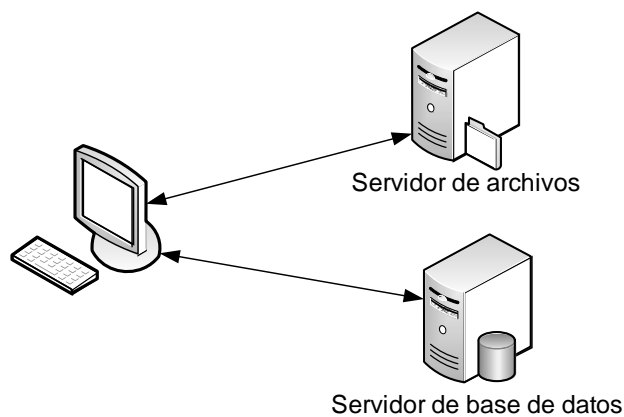
Estos sistemas solían ser acoplados, es decir los componentes de cada capa tienen una dependencia muy alta con los componentes de las otras capas.

A continuación vamos a ver diferentes modelos de arquitecturas distribuidas:

#### Cliente-Servidor

Es un sistema donde el cliente tiene toda la lógica de negocio y acceso a datos y el servidor suele ser únicamente un repositorio de información.

El servidor puede ser un repositorio de archivos, un servidor de base de datos, etc.



**Arquitectura cliente/servidor**

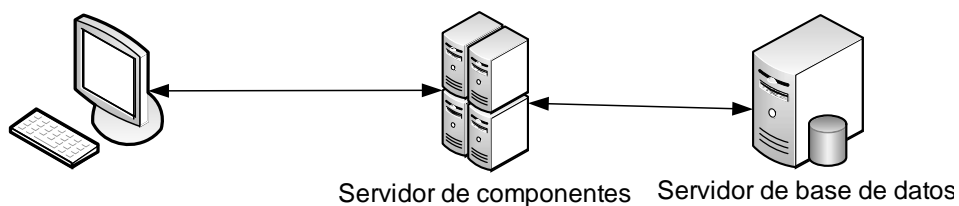
## Arquitectura en tres Niveles (N-Tier)

En una arquitectura de tres capas el cliente se libera del procesamiento de la lógica de negocio y acceso a datos y se convierte en un cliente ligero.

Una aplicación que sigue este modelo está dividida en las siguientes capas:

- **Nivel de presentación:** Suele consistir en una aplicación cliente que únicamente se encarga de implementar el interface con el usuario. Este nivel en un inicio se implementaba como una aplicación Windows, pero ha ido evolucionando de tal forma que actualmente puede ser también una aplicación web.
- **Nivel de componentes de aplicación:** Son componentes relacionados entre sí que se encargan del procesamiento de la lógica de negocio. El nivel de presentación únicamente se comunica con el nivel de negocio para ejecutar las acciones requeridas por el usuario. Normalmente el nivel de negocio está situado en un servidor de componentes de negocio (o varios servidores balanceados). Dentro del servidor de componentes, podemos tener varias capas de componentes (Capa fachada de componentes de negocio, Capa de componentes base de negocio, capa de componentes de acceso a datos, etc.)
- **Nivel de datos:** Son normalmente los servidores de bases de datos, como servidores SQL Server, Oracle, DB2, etc. Pero realmente puede ser cualquier fuente de datos.

La arquitectura en tres niveles es una arquitectura lógica. Físicamente puede haber un número de servidores variable (servidores balanceados, clusters, etc.). La topología de servidores vendrá indicada por los requisitos de rendimiento impuestos por la aplicación.



### Ejemplo de arquitectura de tres capas

En el desarrollo distribuido de hace unos años (finales de los años 90 hasta aproximadamente el año 2002), esta arquitectura se implementaba con **componentes COM** desarrollados en **Visual Basic** y/o **Visual C++**, y el sistema que gestionaba la comunicación distribuida entre el nivel de presentación y los componentes de negocio, era **DCOM**. Así mismo, el sistema transaccional y repositorio de componentes COM fue inicialmente **MTS (Microsoft Transaction Server)** en Windows NT 4.0, y **COM+** en Windows 2000 y posteriores versiones. A estas aplicaciones

## 8 Arquitectura SOA con Tecnología Microsoft

---

3-Tier iniciales, Microsoft las denominó inicialmente aplicaciones **DNA** (*Windows DNA Applications*).

**DCOM** son las siglas de *Distributed Component Object Model*, es una tecnología propietaria de Microsoft que posibilita que dos componentes que se encuentran en servidores distintos se puedan comunicar entre sí como si estuvieran en la misma máquina. Actualmente Microsoft ha abandonado esta tecnología en favor de las tecnologías de conexión remota proporcionadas por Microsoft .NET (Servicios Web, WCF, etc.), las cuales además son interoperables con otras tecnologías como Java.

### **Ventajas de los sistemas distribuidos**

- Una de las principales características de los sistemas distribuidos es la **escalabilidad**. Cada capa de la aplicación puede contener tantos servidores balanceados entre sí como sea necesario.
- Al estar las capas separadas en servidores se aumenta la **conurrencia** y **agilidad** de las aplicaciones dando una respuesta más rápida al cliente.
- Se aumenta la **disponibilidad** de la aplicación. La caída de uno de los servidores suele estar respaldado por otro que suplente al anterior en caso de fallo. De esta forma tenemos sistemas altamente disponibles.
- Aumento de la reutilización de componentes.

### **Desventajas de los sistemas distribuidos**

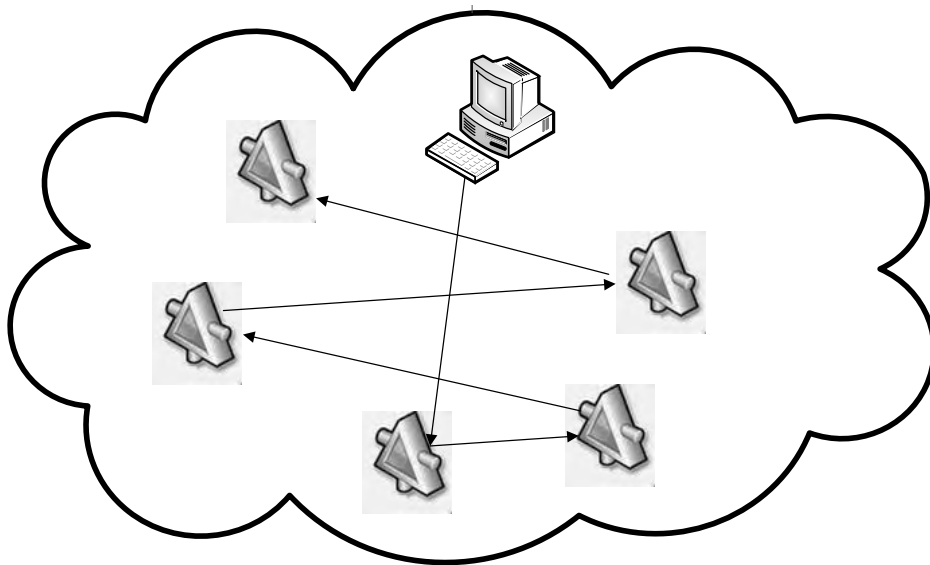
- Los costes iniciales son más altos. El sistema está compuesto por más de un servidor con lo cual el coste de la puesta en producción es mayor que el de las aplicaciones monolíticas. Pero a la larga (con sistemas en producción durante varios años y evolucionando los sistemas), el coste total de propiedad es probablemente más bajo que el de las aplicaciones monolíticas.
- La administración de las aplicaciones suele ser más costosa debido al carácter distribuido del sistema. Se requieren más conocimientos para poder administrar correctamente una aplicación distribuida.
- Se tiene una dependencia muy alta con las redes de comunicación. Un mal funcionamiento de la red afectará negativamente en el rendimiento del sistema distribuido.
- Se debe hacer un mayor hincapié en la seguridad de la información. La distribución de los datos da lugar a problemas potenciales de seguridad que hay que tener en cuenta y mitigar en lo posible.

El desarrollo orientado a servicios (SOA) comparte varios de los principios de las aplicaciones distribuidas orientadas a objetos, como puede ser la encapsulación, la abstracción y las interfaces claramente definidas.

## ¿Qué es SOA?

La arquitectura orientada a servicios es un paradigma para organizar y utilizar capacidades distribuidas que pueden estar controladas bajo diferentes propietarios e implementadas bajo diferentes tecnologías.

La arquitectura orientada a servicios define la base para que un conjunto de servicios independientes puedan colaborar entre sí dando lugar a procesos de negocio más complejos.



**Ejemplo de interacción entre servicios**

Es importante destacar que SOA es un tipo de arquitectura de software, teoría, no está ligado a ninguna tecnología concreta. De hecho puede implementarse un Servicio-SOA con .NET o con Java. En definitiva, se puede aplicar SOA con cualquier tecnología que permita desarrollar servicios interoperables.

Las razones de la aparición de SOA son básicamente las siguientes:

- La Integración entre aplicaciones y plataformas es difícil.
- Existen sistemas heterogéneos (diferentes tecnologías).
- Existen múltiples soluciones de integración, independientes y ajenas unas a otras y normalmente las integraciones han sido siempre muy costosas.

## IO Arquitectura SOA con Tecnología Microsoft

---

Se necesitaba un planteamiento estándar que aporte:

- Arquitectura orientada a servicios.
- Basada en un “bus común de mensajería”.
- Estándares y especificaciones de servicios (WS-\*) para todas las plataformas.

Es primordial que las comunicaciones entre aplicaciones (Servicios) sean independientes de la plataforma (.NET, Java, etc.), de los lenguajes, de los objetos e incluso de los mecanismos de llamada y protocolos de transporte. Estos objetivos son los que conforman SOA (*Service Oriented Architecture*).

SOA ‘ve el mundo’ de una forma distinta, todas las aplicaciones deben ser servicios autónomos donde se definan fronteras explícitas y se asuma la heterogeneidad y colaboren plataformas dispares (por ejemplo comunicar aplicaciones .NET con aplicaciones Java). Para esto, en lo que ya es la ‘implementación de SOA en el mundo real’ es imprescindible hablar un mismo ‘idioma’, es decir, compartir el mismo formato de datos (XML) y los mismos esquemas XML (esquema de mensajes de comunicación SOAP) y basarse en protocolos de comunicación estandar.

La ‘Orientación a Servicios’ se diferencia de la ‘Orientación a Objetos’ primeramente en cómo define el término ‘aplicación’. El ‘Desarrollo Orientado a Objetos’ se centra en aplicaciones que están construidas basadas en librerías de clases interdependientes. SOA, sin embargo, hace hincapié en sistemas que se construyen basándose en un conjunto de servicios autónomos. Esta diferencia tiene un profundo impacto en las asunciones que uno puede hacer sobre el desarrollo.

### ¿Qué es un Servicio?

Un ‘servicio’ es simplemente un programa con el que otros programas interactúan mediante mensajes. Un conjunto de servicios instalados/desplegados sería un sistema. Los servicios individuales se deben de construir de una forma consistente (disponibilidad y estabilidad son cruciales en un servicio). Un sistema agregado/compuesto por varios servicios se debe construir de forma que permita el cambio, el sistema debe adaptarse a la presencia de nuevos servicios que aparezcan a lo largo del tiempo después de que se hubieran desplegado ó instalado los servicios y clientes originales. Y dichos cambios no deben romper la funcionalidad del sistema.

Así pues, un servicio es una unidad de trabajo bien definida, que acepta mensajes de entrada y produce mensajes de salida. La funcionalidad del servicio es expuesta al exterior mediante un interface público bien definido, llamado ‘contrato’.

# feed your brain.<sup>®</sup>



*campus*  
**MVP**

- ✓ Sin tener que desplazarte
- ✓ Sin romper tu ritmo de trabajo
- ✓ Preguntándole a los que más saben

in**fórmate** ya

902 876 475

[www.campusmvp.com](http://www.campusmvp.com)



**Microsoft**  
**GOLD CERTIFIED**  
Partner

Learning Solutions  
Custom Development Solutions



### César de la Torre

Architect Evangelist en Microsoft

*Arquitecto de Software con tecnología Microsoft desde hace 12 años, fué "MVP Connected Systems" antes de entrar a trabajar en Microsoft.*

*Actualmente asesora a arquitectos de software de clientes y partners de Microsoft en enfoques y arranques de proyectos de desarrollo, así como difundir tecnología Microsoft en eventos y conferencias.*

### Roberto González

MVP Biztalk

*Apasionado por la arquitectura de software y el desarrollo de sistemas conectados, tiene 11 años de experiencia como consultor de desarrollo.*

*Actualmente realiza labores de responsable de preventa y formación, además de ser ponente habitual en eventos técnicos de Microsoft.*

El objetivo de este libro es ofrecer un soporte inicial a desarrolladores .NET que, aunque hayan trabajado programando diferentes tipos de aplicaciones, no han llegado a desarrollar aplicaciones con una arquitectura orientada a servicios.

Es realmente difícil localizar un libro en español que sea introductorio y global sobre SOA y que, al mismo tiempo, sea un libro práctico. Esta obra además enseña con ejemplos cómo desarrollar servicios tanto básicos como avanzados y propone recomendaciones de diseño, patrones y mejores prácticas. En definitiva, se pretende ofrecer una ayuda útil y sencilla para quien quiere empezar en las aplicaciones empresariales.

La estructura del libro es por lo tanto muy progresiva e indicada para dicho perfil:

- Se ofrece primero una introducción global de SOA y el desarrollo distribuido.
- A continuación se enseña cómo implementar y consumir servicios web básicos .NET (Servicios Web ASMX). Posteriormente investigamos algunos aspectos más avanzados de los servicios web básicos para seguir con WCF (Windows Communication Foundation), desde sus conceptos básicos hasta aspectos avanzados, seguridad, e incluso un capítulo específico de WCF LOB SDK.
- La partefinal del libro está orientada a las mejores prácticas, recomendaciones y patrones de desarrollo en arquitecturas orientadas a servicios. Se trata de técnicas y consejos que los autores han aprendido a lo largo de años y que además están en muchos casos definidas y contrastadas por la comunidad de desarrolladores. Estas recomendaciones finales y patrones son muy útiles a la hora de tener que implementar un proyecto real.

Finalmente y como colofón, hacemos una breve introducción al concepto de ESB (Enterprise Service Bus), muy importante en grandes implantaciones SOA en empresas con cierto volumen y donde se quiere tener una composición de servicios homogénea y bien estructurada.

NIVEL - Intermedio/Avanzado

**campus**  
**MVP**

[www.krasis.com](http://www.krasis.com)

 **KRASIS**  
PRESS

