FACULTAD DE CIENCIA Y TECNOLOGÍA
INGENIERÍA EN SISTEMAS DE INFORMACION
SEDE ORO VERDE

# “Arquitectura Orientada a Servicios”

Cátedra: Proyecto

Alumno:

- Emanuel Goette

## Integrante

Nombre: Emanuel Goette
Dirección: Los Colibríes 261 – Casa 3 – Oro Verde – Entre Ríos
Teléfono: 0343-154596803
Email: emanuelpeg@yahoo.com.ar

# Índice

# Anteproyecto

## Tema

**"Implementación de Arquitecturas de Software Orientadas a Servicios en la Web 2.0, basada en herramientas Open Source[1]"**

## Introducción

La necesidad de las empresas de integrar la información de clientes remotos con una interfaz rica, de desarrollos rápidos y baratos, de aplicaciones estables y la disminución de los costos de comunicación dio nacimiento a una nueva forma de ver las aplicaciones, conocida como SOA (Service Oriented Architecture). Este estilo arquitectónico concibe una aplicación como un conjunto de servicios, los cuales pueden ser consumidores de otros servicios; de esta forma se puede reutilizar un mismo servicio con varias aplicaciones. El usuario solo tiene una interfaz que consume servicios que forman parte del conjunto de funcionalidades de una aplicación.

En muy pocos casos un sistema no esta integrado o no se comunica con otros sistemas. Un sistema normalmente necesita datos de otro sistema para poder realizar sus procesos diarios. Además es muy común que aplicaciones de una empresa superpongan su funcionamiento, ya que no existen límites bien marcados entre los diferentes componentes que conforman la organización. Los departamentos de TI[2] han resuelto este problema con interfaces costosas que permiten que estos sistemas se comuniquen, limitando su escalabilidad y flexibilidad.

Una respuesta a los problemas citados es una arquitectura flexible orientada a la comunicación, donde las funcionalidades están encapsuladas y son accesibles por todas las aplicaciones y que además presenta una interfaz estándar al consumidor de la funcionalidad. La implementación natural de estas características es el estilo arquitectónico orientado a servicios.

SOA es un estilo arquitectónico que puede ser implementado de varias formas, pero en la actualidad solo algunas se basan en estándares abiertos; estos son los Web Services (WS) SOAP o REST y OSGI. Los Servicios Web permiten que las aplicaciones

[1] **Open Source**: Código abierto, el producto se distribuye con su código fuente, el cual puede ser modificado y recomercializado.
[2] **TI:** Tecnología Informática.

compartan información y que además invoquen funciones de otras aplicaciones independientemente de cómo se hayan creado las aplicaciones, sea cuál sea el sistema operativo o la plataforma en que se ejecutan y cuáles, los dispositivos utilizados para obtener acceso a ellas. Aunque los Servicios Web son independientes entre sí, pueden vincularse y formar un grupo de colaboración para realizar una tarea determinada.

SOA es una arquitectura bastante madura con varios productos en el mercado, los cuales varios son Open Source y gratuitos. Open source significa que se distribuyen con el código fuente dando la libertad de distribuirlo, modificarlo y estudiarlo. Es interesante que un producto puede sea competitivo funcionando con tecnologías abiertas y que permitan tanta libertad. Implementar una arquitectura SOA Open Source trae consigo múltiples ventajas.

La Web experimentó un cambio drástico en los últimos años volcándose a sus usuarios, los cuales tomaron mayor protagonismo. La Web 2.0 es un término que acuña diferentes hechos que describen la nueva web:

- La web se "socializó".
- La web se hizo más interactiva.
- La web brinda ahora mayores canales de comunicación e interrelación.

El fenómeno de las redes sociales, blogs, fotoblogs, wikis entre otros, demuestra que la nueva Web la construyen las personas. Este cambio en la web no es casual, la mayor interactividad lograda con técnicas como Ajax o Ria permitió que las personas la utilicen más. Otro aspecto importante, es que la nueva web esta interrelacionada: los servicios de internet se comunican entre sí todo el tiempo. Web 2.0 se caracteriza porque diferentes sitios web se comunican para brindar mayor funcionalidad e integración. De este modo por ejemplo podemos utilizar una cuenta openID[3] para realizar un comentario en un post de blogger[4].

Una arquitectura de software basada en la comunicación y distribuida como SOA puede ser una herramienta esencial para implementar una web abierta y colaborativa. Una web que busca exponer servicios que se comunican entre sí, brindando funcionalidad

[3] **OpenID:** es un estándar de identificación digital descentralizado, con el que un usuario puede identificarse en una página web a través de una URL (o un XRI en la versión actual) y puede ser verificado por cualquier servidor que soporte el protocolo.
En los sitios que soporten OpenID, los usuarios no tienen que crearse una nueva cuenta de usuario para obtener acceso. En su lugar, solo necesitan disponer de un identificador creado en un servidor que verifique OpenID, llamado proveedor de identidad o IdP.

[4] **Blogger:** es un servicio creado por Pyra Labs para crear y publicar una bitácora en línea. El usuario no tiene que escribir ningún código o instalar programas de servidor o de scripting. Blogger acepta para el alojamiento de las bitácoras su propio servidor (Blogspot) o el servidor que el usuario especifique (FTP o SFTP)

distribuida, e desarrollo de una este tipo de arquitectura más fácil de desarrollar con una arquitectura SOA.

Lo que esta tesis persigue es analizar si es posible desarrollar una aplicación web 2.0 con arquitectura SOA Open Source, distinguiendo las ventajas y desventajas que nos brinda SOA para este tipo de aplicaciones. Para lograr este objetivo se desarrollará una aplicación web 2.0 con arquitectura SOA.

# Problema

¿Es posible desarrollar aplicaciones Web 2.0 con una arquitectura SOA Open Source? ¿Cual son las ventajas y desventajas de la arquitectura SOA para el desarrollo de aplicaciones web 2.0?

# Objetivos

## Generales:

- Definir si es posible implementar el estilo arquitectónico SOA en aplicaciones web 2.0
- Analizar como implementar la arquitectura orientada a servicio.
- Identificar las diferentes ventajas y desventajas de desarrollar una aplicación web 2.0 con una arquitectura SOA.

## Específicos:

- Definir una arquitectura SOA para desarrollar una aplicación Web 2.0 con tecnologías Open Sources.
- Identificar los diferentes frameworks Java que se pueden utilizar para este fin, y definir cual es el más conveniente.
- Implementar una arquitectura SOA con el framework seleccionado.
- Analizar ventajas y desventajas que nos brinda la utilización SOA en el desarrollo de una web 2.0.

# Justificación y Relevancia

Esta tesis se desarrolla sobre la tecnología Java ya que es la que actualmente cuenta con la mayor cantidad de Frameworks de desarrollo SOA, cuenta con herramientas Open Source de gran calidad, y diferentes proveedores, lo cual hace más enriquecedor su análisis.

Existen múltiples Frameworks SOA de diferentes proveedores, pero no hay una clara visión, de la utilidad de implementar SOA para aplicaciones Web 2.0. Esta tesis comprobará si es posible implementar una aplicación web 2.0 con SOA Open Source y marcará las ventajas de desarrollar con SOA.

La tesis a desarrollar será un claro estudio para que el arquitecto de una aplicación Web 2.0 pueda adoptar el estilo arquitectónico SOA conociendo de antemano sus posibilidades, beneficios y precauciones.

# Factibilidad del estudio

El software de desarrollo que se utilizará no necesita grandes requerimientos para ser ejecutado sobre una PC. El mayor desafío se presentará cuando se necesite obtener datos sobre performance, dado que resulta muy difícil probar el comportamiento de la aplicación ante un acceso a grandes volúmenes de datos por parte de muchos usuarios en forma concurrente.

**Factibilidad Económica:** El costo del estudio es relativamente bajo dado que el software que se utilizará para esta investigación es de distribución gratuita, la adquisición de los mismos puede realizarse a través de Internet. Se trabajará solo con frameworks de distribución gratuita y open sources.

**Factibilidad Operativa:** operativamente el desarrollo del proyecto es factible dado que la información sobre el tema es accesible a través de libros e Internet y los frameworks con los que se trabajara son de distribución libre lo que conlleva a su fácil adquisición y la documentación de los mismos. Como se señalo anteriormente el mayor desafío se presentará cuando sea necesario recopilar datos sobre performance, es muy difícil simular escenarios con muchos usuarios trabajando de forma concurrente.

# Marco Teórico

En la Facultad nos enseñan que para comprender una realidad compleja se debe dividirla y de esta manera analizar cada componente y comprenderlo para luego analizar la totalidad de la realidad. Utilizaremos este método para definir la Arquitectura Orientada a Servicios. Primero analicemos el término servicio.

Servicio tradicionalmente se ha utilizado para describir una función de negocio autocontenida, con una interfaz bien definida y estable, que recibe requerimientos de sus clientes. El servicio no depende del contexto de sus clientes y puede ser consumido por varios sistemas sin ser modificado. Los servicios son instalados (o desplegados) una única vez, esto lo diferencia de los componentes que deben ser incluidos dentro del contexto de cada aplicación que requiera de su uso y permanecen disponibles, sin consumir recursos hasta que son invocados.

Propiedades de un servicio:

- Interfaz bien definida.
- Autocontenido.
- No depende del contexto de sus clientes.
- No requiere ser desplegado con cada cliente.

Luego de definir servicio especificaremos arquitectura. Una definición reconocida es la de Clements: “La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones” [PClements].

Una vez aclarada la definición de servicio y arquitectura, podemos avanzar sobre la de SOA. Se podría definir como un estilo arquitectónico que propone modelar la empresa como una colección de servicios expuestos en la red. Nos propone ver a la empresa no como sistemas aislados sino como un todo.

La principal utilidad de los servicios web es que promueven la interoperabilidad entre diferentes plataformas, sistemas y lenguajes. Con servicios web, por ejemplo, sería posible integrar una aplicación Windows desarrollada con Microsoft .NET con una aplicación desarrollada en J2EE[5] desplegada en un servidor de aplicaciones bajo un sistema Linux.

Alrededor de los Web services, que dominan el campo de un estilo SOA más amplio que podría incluir otras opciones, se han generado las controversias que usualmente acompañan a toda idea exitosa. Al principio hasta resultaba difícil encontrar una definición aceptable y consensuada que no fuera una fórmula optimista de mercadotecnia. El grupo de tareas de W3C, por ejemplo, demoró un año y medio en ofrecer la primera definición canónica adecuada, que no viene mal reproducir aquí:

"Un Web service es un sistema de software diseñado para soportar interacción máquina-a-máquina sobre una red. Posee una interfaz descripta en un formato procesable por máquina (específicamente WSDL[6]). Otros sistemas interactúan con el Web service de una manera prescripta por su descripción utilizando mensajes SOAP, típicamente transportados usando HTTP con una serialización en XML en conjunción con otros estándares relacionados a la Web" [WSArchitecture]

Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas. Como muestra la ilustración 1, una aplicación además de implementar sus propios componentes de negocio y datos, también puede reutilizar la funcionalidad de servicios existentes en la red empresarial.

Los componentes del estilo (o sea los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran. Los servicios son las unidades de implementación y diseño.

Como todos los otros estilos, las SOA poseen ventajas y desventajas. Como se trata de una tecnología que está en su pico de expansión, sus virtudes y defectos varían con el tiempo.

[5] **JEE:** Java edición empresarial

[6] **WSDL:** son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web . WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

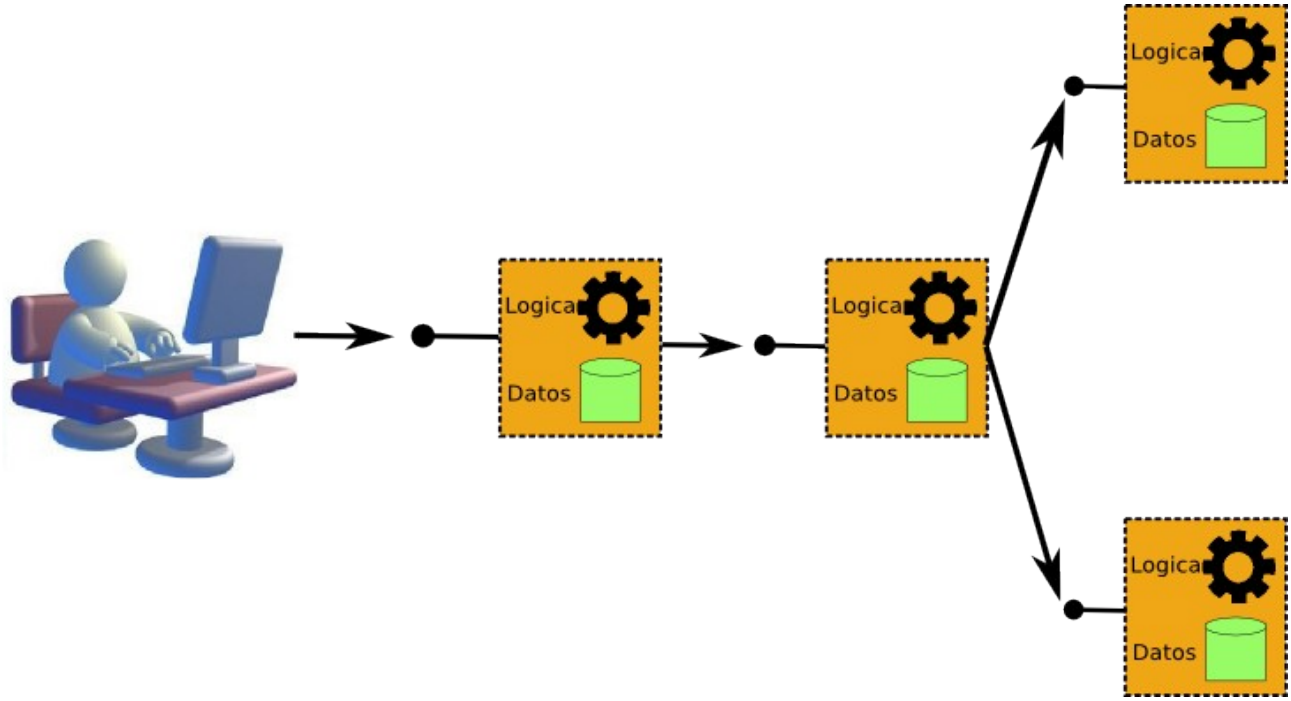


*Ilustración 1: Un sistema sencillo basado en servicios.*

SOA no es un concepto concreto para un área de la empresa, sino que dependiendo del punto de vista que se emplea. Ya que un ejecutivo de negocio lo puede ver como un conjunto de servicios de negocio, un arquitecto de software lo verá como un estilo arquitectónico y el desarrollador un conjunto de estándares, herramientas y tecnologías concretas que permiten llevar a cabo sus tareas. Esta tesis se centrará en la visión del desarrollador buscando las mejores herramientas para el desarrollo de Web Services en Java.

Actualmente en el mercado se encuentran múltiples frameworks para implementar el modelo SOA, algunos basados en el modelo empresarial de Java y otros no. Además hay que analizar la madurez de la solución y la aceptación que tiene en los programadores.

# Hipótesis

Es posible implementar una aplicación Web 2.0 con una arquitectura SOA Open Source.

# Variables

Las variables se pueden englobar en tres:

1. El costo.
2. La performance.
3. La flexibilidad.

En la variable costo vamos a tener: el tiempo de desarrollo, el costo de capacitación y el costo de hardware que conlleva la utilización de estas tecnologías.

En la variable de performance tenemos: tiempo de respuesta a una petición, utilización de recursos.

La flexibilidad engloba la capacidad de adquirir nuevos servicios en el tiempo, la adaptación de la aplicación a los cambios de entorno y la problemática que conlleva el cambio.

# Temas comprendidos en el proyecto

- Conceptos fundamentales sobre el estilo arquitectónico SOA.
- Implementación del modelo SOA Open Source.
- Presentar una conclusión que ayude al lector del proyecto decidir correctamente qué tecnologías le conviene utilizar

# Técnicas e Instrumentos

- Recolección de información tanto de Internet como de libros.

- Sondeo de propuestas para solucionar la problemática de la implementación por medio de recreación de ejemplos; testeando no solo la performance, sino la complejidad de diseño, la flexibilidad y eficacia de los mismos según el caso de estudio.

- Comparar las dimensiones de las variables descriptas a través de un análisis de cada uno de los software seleccionado en el estudio.

# Introducción

A manera de introducción se mencionará como se va a desarrollar la tesis. Este trabajo se dividirá en dos partes, para mayor compresión. En la primera parte se definirá conceptos fundamentales del mundo SOA, para luego poder introducirnos en el estudio de un caso práctico con el cual explicaremos porque es bueno desarrollar aplicaciones Web 2.0 con SOA y herramientas Open Sources.

La teoría tiene como objetivo ubicar al lector en el universo SOA dándole un visión general de los productos y cuando es necesario utilizarlos para poder discernir cuales de estos serán correctos utilizar en el desarrollo web 2.0

En la segunda parte se analizará cuales son las ventajas y desventajas de utilizar SOA en el desarrollo web con herramientas open source a través de un ejemplo práctico. Al finalizar esta sección se deberá tener los elementos necesarios para poder concluir si es beneficioso o no el uso de SOA en el mundo web 2.0.

La conclusión señalará si es posible utilizar SOA Open Source en desarrollos Web 2.0 y además se remarcarán los beneficios y desventajas del uso de la arquitectura SOA en dicho desarrollo. Buscando contrastar las virtudes de SOA con los requerimientos no funcionales de una Web 2.0.

Dicha conclusión será consolidada con el desarrollo de un ejemplo que nos permitirá tomar datos reales de un desarrollo y verificando prácticamente la conclusión.

Luego de analizar los temas comprendidos en esta tesis, comenzaremos analizando de forma teórica, que es SOA y sus cuáles son sus beneficios.

# Desarrollo Teórico

## ¿Que es SOA?

Un sistema debe comunicarse con el mundo exterior para ser útil; normalmente el sistema se comunica con diferentes usuarios pero es muy común que deba comunicarse con otro sistema de la empresa.

Cada compañía usa un impresionante número de sistemas y a la vez deben mantener interfaces de comunicación entre los mismos. Las empresas tienen sistemas viejos, sistemas ERP que por una cuestión de licencia, costo o tecnología es imposible agregar funcionalidad, lo que conlleva a desarrollar un nuevo sistema que provea la funcionalidad faltante en el primero. A pesar de que estos sistemas tienen muy poco en común, resultaba muy difícil justificar su abandono. Al estar tan estrechamente ligados a las operaciones cotidianas del negocio, la sustitución o renovación de estos sistemas resultaba demasiado costosa. Ante esta situación, las compañías encontraron formas distintas de integrar más estrechamente los sistemas, y de facilitar la comunicación entre ellos. Finalmente, todo esto ha llevado a SOA y a la era de la composición.

Normalmente una empresa tiene varios sistemas, los cuales se comunican por medio de interfaces que son difíciles de mantener. Pensar en los web services como una solución para la interoperabilidad entre sistemas, es una solución. Pero podríamos dar un siguiente paso, tomar estos web services, identificarlos según su funcionalidad, organizarlos y coordinarlos para que puedan solucionar requerimientos. Al tener web services orquestados por la arquitectura, la reutilización es óptima y la escalabilidad es inmensa, ya que trabajamos sobre una arquitectura distribuida.

“La **Arquitectura Orientada a Servicios** (en inglés **S**ervice **O**riented **A**rchitecture), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio.

Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios, lo cual facilita la interacción entre diferentes sistemas propios o de terceros.”[WikiSOA]

SOA es una forma de ver los sistemas, normalmente un sistema se ve como un conjunto de funcionalidades las cuales se encuentran bajo una interfaz y además agrupadas en un sistema, acopladas al mismo. Con SOA los sistemas son conjuntos de funcionalidades las cuales tienen una definición y están publicadas para ser accedidas, el conjunto de funcionalidad son orquestadas por un software que permite decidir el flujo de ejecución.

"SOA define las siguientes capas de software:

- **Aplicaciones básicas** - Sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad;
- **De exposición de funcionalidades** - Donde las funcionalidades de la capa aplicativa son expuestas en forma de servicios (servicios web);
- **De integración de servicios** - Facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración;
- **De composición de procesos** - Que define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio;
- **De entrega** - donde los servicios son desplegados a los usuarios finales.

SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación."[WikiSOA]

# ¿Cuándo implementar SOA?

Según [AdopcionSOA] SOA es una arquitectura que cambia totalmente el departamento de IT y también la empresa, migrar a una arquitectura SOA, no es fácil y requiere esfuerzo humano y técnico. Siempre que se tenga el presupuesto para migrar es bueno hacerlo. Ya que los beneficios de usar una arquitectura SOA, son muchos.

Para nombrar algunos:

- No existirán funcionalidades replicada en varios sistemas.
- No existirán funcionalidades o procesos de integración entre sistemas.
- Mayor flexibilidad e integración.
- Bajo acoplamiento.

- Integración basada en estándares.

# ¿Que define la Arquitectura SOA?

La arquitectura SOA define :

- Los servicios.
- Como localizar un servicio.
- Como conseguir que se comuniquen diferentes servicios.
- Como encajar diferentes servicios para que funcionen como un sistema.

En una SOA, los servicios se encuentran documentados en un repositorio denominado registro, se ensamblan mediante las llamadas aplicaciones compuestas, y el plano que le sirve de guía es lo que se conoce como esquema global[7] de la SOA.

Para mayor facilidad de los conceptos que conllevan el estilo arquitectónico SOA, vamos a analizar primero los servicios y luego las herramientas que nos permiten administrarlos.

# Servicios.

Los servicios son funcionalidad necesaria para la empresa, es decir son funcionalidad expuesta a otros servicios, sistemas o una interfaz gráfica. Los cuales fueron concebidos para solucionar un requerimiento. Los servicios son los ladrillos de nuestra arquitectura SOA. Ellos pueden contener lógica de negocio, acceso a bases de datos, accesos a otros servicios, etc.

Los servicios están descriptos en una interfaz que indica como utilizarlos. Pero los servicios nunca exponen como funcionan; son una caja negra. Lo que permite cambiar un servicio fácilmente por otro, favoreciendo la mantenibilidad y agilidad de desarrollo. Los servicios exponen como se utilizan pero no exponen como funcionan.

Los servicios pueden acoplarse para formar nuevos servicios, que solucionan nuevos requerimientos, esto ayuda a que no se duplique el código. Los servicios son atómicos y sin estado lo que permite distribuirlos en diferentes maquinas favoreciendo la escalabilidad de los sistemas.

[7] **Esquema global:** de una SOA debe indicar el estado objetivo. Esto significa que debe ofrecer una imagen completa de la implementación de la SOA una vez que esté finalizada.

# Gobierno SOA

Los servicios por si solos son incompletos, necesitan una arquitectura que los mantenga y soporte sus ventajas. Por ejemplo a la hora de desarrollar un nuevo servicio ¿cómo se si no existe ya? O ¿existen servicios que pueden ayudar al nuevo servicio? Por lo tanto debo tener un registro donde registrar mis servicios, en el cual puedo buscar los servicios ya existentes.

El gobierno SOA es el conjunto de roles, políticas y procedimientos que sirven de guía para la adopción de la SOA. Al implementar los componentes tecnológicos de gobierno, está creando la infraestructura para soportar y aplicar estos roles, políticas y procedimientos en toda su SOA.

Luego de desarrollar los servicios hay que gestionarlos, la gestión de los servicios se le llama gobierno SOA.

"Gobierno de SOA es una ampliación del gobierno de IT centrada en el ciclo de vida[8] de los servicios y en las aplicaciones compuestas en una arquitectura orientada a servicios (SOA) de una organización. La función del gobierno de SOA es definir:

- Derechos para tomar decisiones para el desarrollo, el despliegue y la gestión de nuevos servicios.

- La supervisión y notificación de procesos para capturar y comunicar los resultados del gobierno. Debido a que las aplicaciones SOA están intrínsecamente fragmentadas, introducen nuevos retos de gobierno. No obstante, con políticas, principios, estándares y procedimientos adecuados, los empresarios pueden darse cuenta de todas las ventajas que ofrece la arquitectura orientada a servicios. Una plataforma de gobierno de SOA eficaz no sólo ayuda a los equipos de IT y negocio a identificar mejor qué proyectos contribuyen más a los objetivos empresariales, sino que también dotan de más facilidades a los empleados para trabajar y colaborar de forma más eficaz definiendo claramente sus roles y sus responsabilidades." [IBMGobiernoSOA]

El Gobierno SOA pretende dotar de los mecanismos de control, procesos, procedimientos y métodos probados en la práctica para garantizar el orden en las decisiones que se tomen en una iniciativa SOA, evitando el caos en cualquier proyecto

[8] **Ciclo de vida:** Los ciclos de vida definen las fases por las que pasa un servicio mientras está activo en la SOA. Definir un ciclo de vida como éste para su organización es una de las primeras actividades de gobierno que debe acometer. El registro/repositorio le debería permitir modelar y monitorizar explícitamente este ciclo de vida.

SOA que se plantee, logrando la efectividad y agilidad esperada en la transición hacia SOA.

El Gobierno se refiere a los procesos que una empresa establece para garantizar que las acciones realizadas se adhieren a las mejores prácticas, normativas, estándares y principios de arquitectura, con objeto de gobernar la adopción e implementación de SOA.

La ausencia del Gobierno SOA puede desembocar en los siguientes problemas:

- El Programa SOA entrega resultados inconsistentes.
- Crecimiento caótico a nivel de infraestructura y servicios.
- Servicios con funcionalidad redundante.
- Disponer de servicios que no se pueden reutilizar en la organización.
- Inconsistencia en la identificación, diseño y uso de los servicios.
- No se definen métricas para cuantificar el éxito.
- Ausencia de coordinación entre proyectos.

Según [AdopcionSOA] la definición de una Estrategia de Gobierno SOA proporciona la coherencia necesaria para todas las acciones que se lleven a cabo en un Plan de Adopción SOA, siguiendo un enfoque incremental de adopción en distintas fases para reducir riesgos, y proporcionando un modelo de gobierno que constituya un marco de referencia para las decisiones que se tomen.

Por lo tanto podríamos definir a SOA como la suma de servicios y gobierno SOA.

Como se puede leer en [OpenSourceSOA] el Gobierno SOA comprende varias herramientas que permiten gobernar los servicios:

- Registro de servicios, permite registrar nuestros servicios y consultar servicios existentes.
- Bus de servicios o ESB (Enterprise Service Bus) facilita la comunicación entre diferentes servicios.
- Orquestación y coreografía, herramienta que permite definir la interoperabilidad de los servicios.
- CEP (Complex Event Processing) y Procesador de flujos de eventos o ESP(Event Stream Processing) son herramientas para diseñar, gestionar y

monitorizar los eventos que fluyen a través de un entorno orientado a EDA[9] dado.

- Integración de aplicaciones empresariales o EAI (Enterprise Application Integration) facilita la integración con aplicaciones o bases de datos de una empresa.

A la vez existen muchos frameworks que nos permiten desarrollar servicios SOA, con mayor flexibilidad y rapidez. Permitiendo obtener más fácilmente los beneficios del uso SOA y a la vez preparados para ser Gobernados por el gobierno SOA. En la tesis nos centraremos en el estándar de OASIS[10], SCA (Service Component Architecture). SCA es un estándar que simplifica el desarrollo SOA basándose en el concepto de componentes. Nos centraremos en esta especificación dado que brinda gran facilidad en el desarrollo, además promueve el uso de las mejores prácticas.

Como describe [AdopcionSOA] no necesariamente se deben tener todas las herramientas descriptas para conformar el gobierno SOA. Se deben adoptar las herramientas que satisfagan las necesidades propias de la empresa que adopta SOA. A continuación se describirá las herramientas SOA.

# Registro o repositorio

Como se indica en [AdopcionSOA] sólo se puede gobernar lo que se ve, por lo tanto, el primer paso es crear un único catálogo maestro en el que estén visibles, para todas las partes interesadas, los elementos más importantes de su SOA. El registro/repositorio se ha erigido en el estándar para la creación de este tipo de sistema de registros.

El registro es donde se publican los servicios y es donde se debe consultar los servicios existentes para poder consumirlos.

El registro puede ser desde una base de datos LDAP[11] a una entrada en wiki de la empresa, dependiendo de los requerimientos de la empresa. Pero siempre debe tener al menos esta información:

[9] **EDA:** Event-Driven Architecture

[10] **OASIS:** acrónimo de Organization for the Advancement of Structured Information Standards es un consorcio internacional sin fines de lucro que orienta el desarrollo, la convergencia y la adopción de los estándares de comercio electrónico y servicios web.

[11] **LDAP:** *Lightweight Directory Access Protocol*, Protocolo Ligero de Acceso a Directorios, es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. LDAP también es considerado una base de datos (aunque su sistema de almacenamiento puede ser diferente) a la que pueden realizarse consultas. LDAP es un protocolo de acceso unificado a un conjunto de información sobre una red.

- Punto final del servicio (donde esta publicado).
- Descripción del servicio.
- Ubicación WSDL si usamos servicios SOAP o como se utiliza el servicio.
- Numero de revisión y versión.

# ESB (Enterprise Service Bus)

Si en nuestra organización los sistemas se comunican entre si por medio de un patrón punto a punto, los canales de comunicación se multiplicarían y a la vez se debería implementar procesos de traducción protocolos para que cada sistema pueda comunicarse en el protocolo que fue concedido, veamos un ejemplo:

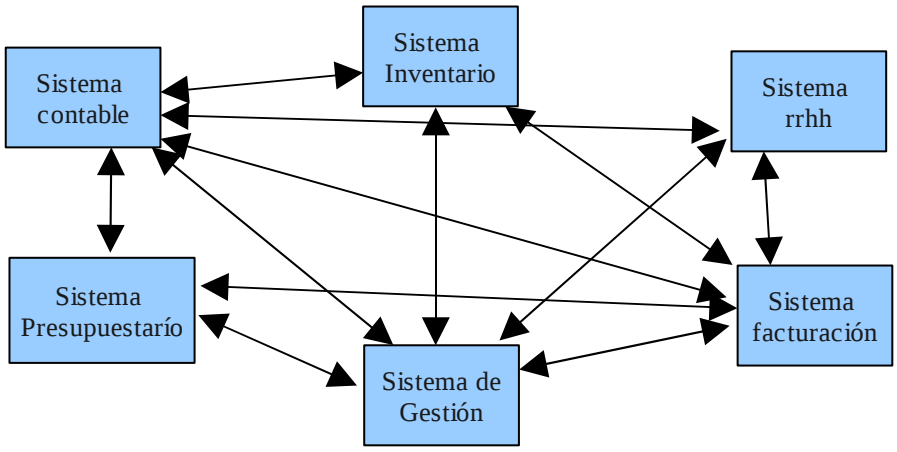


*Ilustración 2: Ejemplo de comunicación punto a punto entre aplicaciones*

Las comunicaciones no están centralizadas lo que trae mayor canal de comunicación y como cada aplicación habla un idioma diferente: REST, SOAP, RMI, etc. las funcionalidades de traducción se duplican.

Por lo tanto, si tuviéramos las comunicaciones en un Bus simplificaremos la comunicación y además no se duplicaría el código de traducción ya que se encuentra centralizado, podemos ver gráficamente un ejemplo del uso de un Bus en la ilustración 3

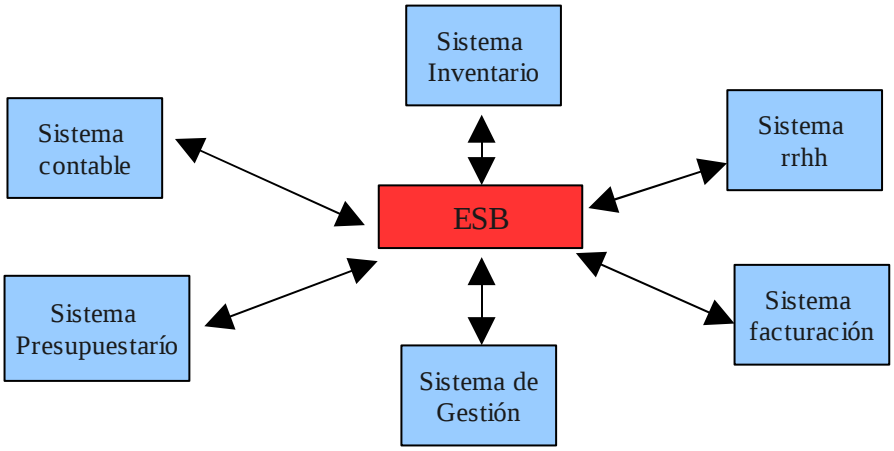


*Ilustración 3: Ejemplo de uso de ESB*

"En una arquitectura tan compleja, el ESB representa el elemento de software que media entre las aplicaciones empresariales y permite la comunicación entre ellas. Idealmente el ESB tendría que ser capaz de sustituir todo contacto directo con las aplicaciones en el bus, de modo que toda la comunicación tenga lugar a través del bus. Para lograr este objetivo, el bus debe encapsular la funcionalidad que ofrecen las aplicaciones que lo componen de un modo significativo. Esto sucede normalmente con la implantación de un modelo de mensaje de empresa. El modelo de mensajes define un conjunto de mensajes normalizado que el ESB recibe y transmite. Cuando un ESB recibe un mensaje, lo encamina hacia la aplicación apropiada. A menudo sucede que, como esa aplicación se ha desarrollado sin el mismo modelo de mensajes, el ESB tendrá que transformar el mensaje a un formato de compatibilidad (*legacy format*) que la aplicación sea capaz de interpretar. Un "adaptador" de software lleva a cabo la tarea de efectuar estas transformaciones (al igual que lo hace un adaptador físico). No hay acuerdo en si se debe considerar este adaptador como constituyente del ESB o no.

Los ESB se basan en la conexión precisa de un modelo de mensajes de empresa y la funcionalidad ofrecida por las aplicaciones. Si el modelo de mensajes no encapsula completamente la funcionalidad de las aplicaciones, entonces otras aplicaciones que desean esa funcionalidad pueden tener que rodear el bus e invocar directamente a las aplicaciones no emparejadas. Esto supone violar todos los principios señalados más arriba y desprecia muchas de las ventajas de la utilización de un ESB." [WikiESB]

Un bus de servicios empresariales o ESB (Enterprise Service Bus) es una buena elección para la capacitación de servicios, si la aplicación que está intentando habilitar proporciona una interfaz para conectarla a otros sistemas. Un buen ESB ofrece todas las herramientas que necesita para crear servicios XML que aprovechen ese API:

- Compatibilidad con varios protocolos: Los ESB implementan un gran abanico de protocolos, especialmente algunos ya pasados de moda como por ejemplo: RPC[12]. Un buen ESB se ocupa de todos los detalles para poder utilizar un protocolo, de forma prácticamente transparente.

- Compatibilidad con diferentes patrones de comunicación: El medio más común que utiliza un ESB para comunicarse con una aplicación es mediante el patrón de petición/respuesta (request/reply). Según este patrón, el ESB envía una consulta a la aplicación correspondiente mediante un protocolo compatible y la aplicación le remite inmediatamente la respuesta. Pero muchos sistemas de misión crítica utilizan patrones de comunicación más sofisticados y orientados a mensajes, como 'publicación/suscripción' o 'envía y olvida'. Un buen ESB consigue que la conexión a un sistema mediante patrones avanzados de comunicación sea un proceso sencillo, al igual que la combinación y compatibilización de patrones según se necesitan.

- Compatibilidad con diferentes formatos de mensajes: Los ESB son también muy buenos en la traducción de mensajes XML a un lenguaje comprensible para sus aplicaciones o viceversa. No importa que se trate de un lenguaje MIME, sólo texto o archivos planos. El ESB ejecuta todas las traducciones y transformaciones necesarias para convertir hacia o desde XML.

- Adaptadores: Los ESB pueden gestionar todos los detalles necesarios para conectar las aplicaciones existentes. Los mejores ESB ocultan la compleja labor de conectar una aplicación tras interfaces comunes y coherentes denominadas adaptadores. Estos reducen enormemente la curva de aprendizaje de los desarrolladores de servicios, que en lugar de ocuparse de las complejas conexiones entre diferentes sistemas, pueden centrarse en exponer las funcionalidades existentes como servicios.

Contar con un ESB en una empresa es muy importante dado que contribuye a la sana convivencia de diferentes aplicaciones que se encuentran implementadas en diferentes plataformas y hablan diferentes idiomas. Podemos ver un ejemplo en la Ilustración 4.

[12] **RPC:** del inglés *Remote Procedure Call*, **Llamada a Procedimiento Remoto,** es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos. El protocolo es un gran avance sobre los sockets usados hasta el momento. De esta manera el programador no tenía que estar pendiente de las comunicaciones, estando éstas encapsuladas dentro de las RPC.

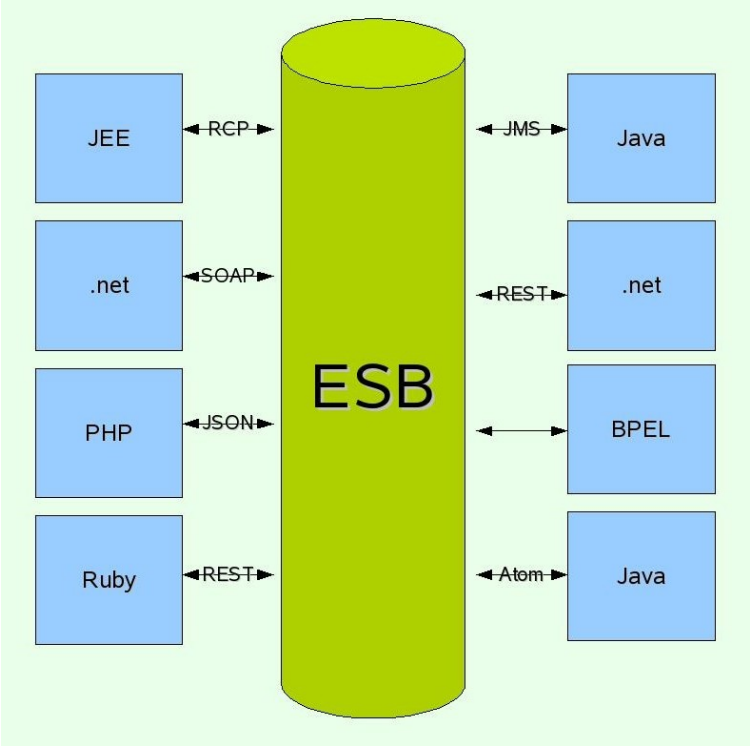


*Ilustración 4: Ejemplo de uso de ESB entre diferentes plataformas*

# Orquestación y Coreografía.

"La integración se debe llevar a cabo mediante un mecanismo que permita que los servicios cooperen entre ellos. Para ello comúnmente se utilizan dos términos: La orquestación y la coreografía. Hablamos de orquestación de servicios cuando es controlado por una única unidad, es decir un cliente y un servicio establecen un acuerdo con respecto al transporte de mensajes y al contenido. Un proceso es una coreografía de servicios cuando las colaboraciones son definidas entre cualquier tipo de aplicaciones." [W3CCDL]

Como indica [BPMandSOA] la coreografía brinda reglas que revelan como múltiples componentes pueden colaborar entre pares, lo que denominamos peer-to-peer, y de que forma o en que secuencia. Esta colaboración se realiza mediante el intercambio de mensajes. Una *coreografía* describe la colaboración entre más de un servicio para lograr un objetivo común. Por lo tanto no se centra en un servicio en particular, pero si en un objetivo que tiene que cumplir el flujo de proceso. Para ello, la lógica de control es distribuida sobre los servicios participantes. Para diseñar una coreografía, se debe describir primero las interacciones que los servicios tienen que realizar entre sí para cumplir el objetivo y las relaciones que existen entre estas interacciones. Una *coreografía*

no describe las acciones que son realizadas internamente por el servicio principal para realizar el flujo de procesos.

Si alguna vez pudo apreciar una presentación de ballet, puede haberse percatado como cada integrante del equipo (servicios) inicia y finaliza un conjunto de movimientos (flujo de proceso), estos movimientos siguen una trilla (interacciones entre los servicios) musical determinada.

Una *orquestación* describe el comportamiento que el servicio principal implementa para realizar un flujo de proceso. Esto quiere decir, que el enfoque se centra sobre un servicio en particular, y la lógica de control es centralizada sobre el servicio principal el cual implementa el comportamiento. Para diseñar una *orquestación*, se describe las interacciones que el servicio principal tiene con las otras partes (servicios, adaptadores, interfaces, etc.) y las acciones que el servicio principal realiza internamente para realizar el flujo de proceso. Una *orquestación* es ejecutada por una máquina de orquestación. Por lo que es llamado proceso ejecutable.

Para entender mejor la definición de *orquestación*, imagine como un maestro de una orquesta sinfónica dirige y controla a cada uno de los integrantes (servicios) de la orquesta con el fin de poder lograr un objetivo común (un flujo de proceso), que es el de poder establecer una melodía armoniosa basada en un conjunto de especificaciones (objetivo principal y interrelaciones entre servicios) que contienen las notas, que instrumentos deben tocarse, que notas deben tocar cada uno de esos instrumentos y cuando deben iniciar o detener sus melodías.

Coreografía nos permite especificar las reglas de unión y trabajo colaborativo (entendiendo por colaboración, una función/es de negocio surgidas de la interacción cooperativa de múltiples actores). Es lo que normalmente se entiende por un proceso de negocio global donde se modela el estado de negocio de diversos servicios web.

La orquestación es un mecanismo para el intercambio de mensajes desde una visión más detallada a través de un proceso (un flujo de control). La Orquestación podría verse como la ejecución de un proceso de negocio definido en BPM o BPEL y que puede ser ejecutado por un motor BPM o BPEL.

La coreografía se define mediante WS-CDL y la orquestación puede definirse con BPM o BPEL veamos cada uno de estos lenguajes.

## WS-CDL

Cuando hablamos de coreografía de Servicios Web se debe mencionar a WS-CDL (Web Services Choreography Description Language) y por tanto de colaboración entre actores; es decir, de interacciones entre servicios web. Este lenguaje, basado en XML, permite lograr interacción entre servicios Web. Dicha interacción es independiente del lenguaje o de la plataforma utilizada.

La coreografía es una visión más abstracta y descriptiva de los actores que intercambian mensajes para ejecutar varios procesos particulares (varios flujos de control). WS-CDL tiene como propósito definir la interoperabilidad necesaria para crear un sistema compuesto por servicios web.

## BPM

Como indica [IntroduccionBPM] BPM (Business Process Management) podría ser definido como un Flujo de trabajo o workflow[13], es una forma de representar los procesos, los cuales son seguidos por el sistema. Permitiendo en tiempo de ejecución modificar el workflow y que se aplique estos cambios sin volver a compilar ni a bajar o subir la aplicación.

BPM marca al sistema el orden de ejecución de procesos dando la flexibilidad para poder modificar este orden cuando se lo desee. Esto permite centrarse en la lógica de negocio dando el poder de decisión al usuario final el cual por medio de un editor puede modificar el workflow como crea más conveniente. Se centra en el negocio.

BPM es un concepto muy sencillo. Es un conjunto de métodos, herramientas y tecnologías utilizados para diseñar, representar, analizar y controlar procesos de negocio operacionales; un enfoque centrado en los procesos para mejorar el rendimiento que combina las tecnologías de la información con metodologías de proceso y gobierno.

El objetivo de una solución para la gestión de procesos de negocio es proporcionar, dentro de las TI, implementaciones automatizadas de procesos de negocio de la vida real, como por ejemplo los procesos de pedidos y cobros, o de gestión de reclamaciones. Combinada con una SOA, esta forma de proporcionar funcionalidades de TI, con una visión orientada a procesos.

La tecnología BPM incluye lo necesario para diseñar, representar, analizar y controlar los procesos de negocio operacionales:

[13] **Workflow:** El Flujo de trabajo es el estudio de los aspectos operacionales de una actividad de trabajo: cómo se estructuran las tareas, cómo se realizan, cuál es su orden correlativo, cómo se sincronizan, cómo fluye la información que soporta las tareas y cómo se le hace seguimiento al cumplimiento de las tareas. Generalmente los problemas de flujo de trabajo se modelan con redes de Petri

- El diseño y modelado de procesos posibilitan que, de forma fácil y rigurosa, pueda definir procesos que abarcan cadenas de valor y coordinar los roles y comportamientos de todas las personas, sistemas y otros recursos necesarios.

- La integración le permite incluir en los procesos de negocio cualquier sistema de información, sistema de control, fuente de datos o cualquier otra tecnología. La arquitectura orientada a servicios (SOA) lo hace más rápido y fácil.

- La ejecución convierte de forma directa los modelos en acción del mundo real, coordinando los procesos en tiempo real.

- La supervisión de la actividad de negocio (BAM) realiza el seguimiento del rendimiento de los procesos mientras suceden, controlando muchos indicadores, mostrando las métricas de los procesos y tendencias clave y prediciendo futuros comportamientos.

- El control le permite responder a eventos en los procesos de acuerdo a las circunstancias, como cambio en las reglas, notificaciones, excepciones y transferencia de incidentes a un nivel superior.

Para soportar esta estrategia es necesario contar con un conjunto de herramientas que den el soporte necesario para cumplir con el ciclo de vida de *BPM*. Este conjunto de herramientas son llamadas Business Process Management System y con ellas se construyen aplicaciones *BPM*.

BPM no esta acoplado al mundo SOA, se puede controlar procesos de una aplicación usando BPM sin tener servicios web. Pero BPM se puede utilizar para modelar flujo de trabajo llamando servicios web. Permitiendo una gran flexibilidad el flujo de trabajo no depende de una aplicación sino que esta constituido por servicios referenciados por una URI. El motor BPM no esta acoplado a implementaciones sino que conoce solo interfaces dadas por WSDLs o otros descriptores desacoplando el motor BPM de los servicios. Además se puede extender el motor BPM para que ofrezca información del workflow por servicios web. Proporcionando funcionalidades completas por servicios web.

## BPEL

Business Process Execution Language (lenguaje de ejecución de procesos de negocio), se trata de un lenguaje XML para la especificación de procesos de negocio ejecutables, aplicado principalmente a la orquestación de los servicios web. BPEL es un subconjunto de BPM. Definido para representar workflows de servicios web.

La interacción entre servicios web puede ser descripta de dos formas: procesos de negocio ejecutable y procesos de negocio abstracto.

- Proceso de negocio ejecutable: es el comportamiento real de un participante de una interacción.

- Proceso de negocio abstracto: es el proceso que no está destinados a ser ejecutados, puede ocultar algunos de los detalles necesarios de las operaciones concretas. Los Procesos abstractos desempeñan una función descriptiva, con más de un caso de uso posible, incluyendo el comportamiento observable y plantilla de proceso. Un proceso abstracto que incluye información como cuándo es mejor esperar para mensajes, cuándo enviar mensajes, cuándo compensar las transacciones fallidas, etc.

BPEL provee un lenguaje para la especificación de Procesos de negocios Ejecutables y Abstractos. De este modo, se extiende el modelo de servicios Web y la posibilidad de interacción para apoyar las transacciones comerciales. WS-BPEL define un modelo de integración interoperable que debería facilitar la expansión de la integración de procesos automatizados, tanto dentro como entre las empresas.

Bpel fue diseñado con los siguientes objetivos:

1. Definir procesos de negocio que interactúan con entidades externas a través de operaciones de Web services definidas en archivos WSDL, y que se manifiestan en los servicios Web definen utilizando WSDL.

2. Esta basado en xml. No define una representación gráfica de procesos, ni provee algún particular metodología de diseño.

3. Definir una serie de conceptos de orquestación de servicios Web que pretenden ser usados por vistas internas o externas de un proceso de negocio.

4. Proveer sistemas de control jerárquicos y de estilo grafo, que permitan que su uso sea lo más fusionado e inconsútil posible. Esto reduciría la fragmentación del espacio del modelado de procesos.

5. Proveer funciones de manipulación simple de datos, requeridas para definir datos de procesos y flujos de control.

6. Soportar un método de identificación de instancias de procesos que permita la definición de identificadores de instancias a nivel de mensajes de aplicaciones. Los identificadores de instancias deben ser definidos por socios y pueden cambiar.

7. Brindar la posibilidad de la creación y terminación implícitas de instancias de procesos, como un mecanismo básico de ciclo de vida. Operaciones avanzadas de ciclo de vida como por ejemplo "suspender" y "continuar" pueden agregarse en futuras versiones para mejorar el manejo del ciclo de vida.

8. Definir un modelo de transacción de largo plazo que se base en técnicas probadas tales como acciones de compensación y ámbito, de tal manera a brindar recuperación a fallos para partes de procesos de negocios de largo plazo.

9. Usar servicios Web como modelo para la descomposición y ensamblaje de procesos.

10. Construir sobre estándares de servicios Web (aprobados y propuestos) tanto como sea posible, de manera modular y extensible.

BPEL es una herramienta concebida para diseñar workflows basados en web services, esto lo hace una herramienta más apta que BPM para la orquestación de servicios SOA. BPM es más amplio y menos específico para SOA por estas razones nace BPEL. Un lenguaje más específico que suple las deficiencias que tiene BPM.

# CEP (Complex Event Processing) y ESP (Event Stream Processing)

Las tecnologías emergentes como el Procesamiento de Eventos Complejos (Complex Event Processing o CEP) y el Procesamiento de Flujos de Eventos (Event Stream Processing o ESP) han comenzado a ocuparse de problemas de mayor complejidad que no podían resolverse con el procesamiento de eventos tradicional. Por ejemplo, con CEP se puede “analizar, correlacionar y resumir eventos de bajo nivel en eventos de más alto nivel adecuados para informar a las personas en términos humanos o para desencadenar procesos automáticos.” [Complexevents] CEP y ESP emplean técnicas como la detección de patrones complejos en numerosos eventos, utilizando algoritmos de procesamiento de reglas para la correlación y abstracción de eventos, empleando jerarquías de eventos y relaciones entre los eventos. Los análisis de causalidad, membrecía, oportunidad y procesos impulsados por eventos son capacidades centrales de estas tecnologías.

ESP y CEP se consideran parte de una tendencia más extensa llamada EDA (event-Driven Architecture) EDA es un estilo arquitectónico, que esta centrado en comunicaciones asíncronas basadas en eventos. Es totalmente complementario a SOA y utiliza mensajes asíncronos en lugar de llamadas a funciones al estilo RPC, para realizar computación distribuida. Un evento es simplemente la acción de algo que está ocurriendo, bien sea un pedido, un aviso de entrega o la finalización del trabajo de un empleado. El sistema que registra el evento (también llamado sensor) genera un objeto de eventos que se envía mediante una notificación. El consumidor de la notificación o receptor puede ser otro sistema que por su parte, utilice el evento para iniciar alguna acción como respuesta. Aquí es donde el concepto de ESP entra en juego.

Como indica [**OpenSourceSOA**] ESP permite analizar los procesos e identificar eventos inusuales, lo que permite a raíz de estos eventos informarlos o ejecutar algún proceso. Por ejemplo una persona que saca del cajero un monto inusual de dinero el sistema podría enviar un e-mail al cliente notificando la extracción. Otro ejemplo podría ser: un servicio demora más de lo normal esto puede generar un evento el cual sea notificado a un servicio que informe al administrador.

Implementar ESP en una arquitectura SOA nos facilita su mantenimiento y permite manejar eventos complejos de una forma fácil.

# EAI (Enterprise Application Integration)

EAI (*Enterprise Application Integration*, traducido al español como *integración de aplicaciones empresariales*) es definido como el uso de software y sistemas de computadora para integrar un conjunto de aplicaciones.

El propósito de la **EAI** es lograr la interoperabilidad y organización del flujo de información entre aplicaciones heterogéneas, por lo tanto, asegurar la comunicación entre las distintas aplicaciones y formar el sistema de información de la empresa, incluso de los clientes, socios o proveedores.

Por lo tanto, y en primer lugar, un proyecto de EAI implica implementar una arquitectura bajo la cual las distintas aplicaciones se comuniquen entre sí. En consecuencia, esto conlleva el desarrollo de **conectores** (*middleware*) que posibilitan la interfaz de aplicaciones mediante el uso de distintos protocolos de comunicación (por lo general exclusivos).

Sin embargo, el proyecto de *EAI* va más allá de la interoperabilidad de las aplicaciones: ofrece la posibilidad de definir un workflow entre las aplicaciones; así, representa una alternativa a la ERP[14] con un enfoque más modular.

No obstante, la EAI todavía presenta limitaciones relacionadas con la rigidez de los sistemas heredados (en inglés *legacy*), porque se debe modificar el middleware cuando hay cambios importantes en las aplicaciones.

Según [WikiEAI] cuando los sistemas no pueden compartir su información efectivamente se crean cuellos de botella que requieren de la intervención humana en la forma de toma de decisiones o en el ingreso mismo de la información. Con una arquitectura EAI correctamente implementada, las organizaciones pueden enfocar la mayoría de sus esfuerzos en la creación de competencias que generen valor en lugar de enfocarse en la coordinación de labores operativas.

[14]**ERP:** Los sistemas de planificación de recursos empresariales, o ERP (por sus siglas en inglés, *Enterprise resource planning*) son sistemas de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía comprometida en la producción de bienes o servicios.
La Planificación de Recursos Empresariales es un término derivado de la Planificación de Recursos de Manufactura (MRPII) y seguido de la Planificación de Requerimientos de Material (MRP). Los sistemas ERP típicamente manejan la producción, logística, distribución, inventario, envíos, facturas y contabilidad de la compañía. Sin embargo, la Planificación de Recursos Empresariales o el software ERP puede intervenir en el control de muchas actividades de negocios como ventas, entregas, pagos, producción, administración de inventarios, calidad de administración y la administración de recursos humanos.

Por años los sistemas se limitaron a una pobre comunicación con otros sistemas de una organización. Uno de los retos de una organización hoy día es brindar información completa y actualizada en el momento preciso, lo que conlleva a comunicar a sistemas que pueden estar hace muchos años en la empresa.

EAI, como una disciplina, busca solventar muchos de esos problemas, así como crear nuevos paradigmas para ciertamente mejorar las organizaciones, tratando de trascender el objetivo de conectar las aplicaciones individuales para buscar ser un mecanismo de incrementar el conocimiento de la organización y crear ventajas competitivas futuras a la empresa.

La integración de aplicaciones de empresa ha incrementado su importancia porque la computación en las empresas frecuentemente toma la forma de islas de información. Esto ocasiona que el valor de los sistemas individuales no sea aprovechado al máximo debido a su mismo aislamiento.

Si la integración se aplica sin seguir un enfoque estructurado de EAI, las conexiones punto a punto crecen al interior de la organización resultando en una masa imberbe que es difícil de mantener. Esto se denota normalmente como el espagueti, en alusión al equivalente en programación: el código espagueti.

EAI puede ser usado con diferentes fines:

- Integración de datos (información): asegurando que la información en varios sistemas es consistente. Esto también se conoce como EII (Enterprise Information Integration).
- Integración de procesos: enlace de los procesos de negocios entre diferentes aplicaciones.
- Independencia de proveedor: extrayendo las políticas o reglas del negocio de las aplicaciones e implementándolas en un sistema EAI, de forma que cualquiera de las aplicaciones usadas pueda ser cambiada sin que dichas reglas de negocio deban ser reimplementadas.
- Facade común: Un sistema EAI puede actuar como el front-end de un cúmulo de aplicaciones, proporcionando una interfaz de acceso única y consistente a esas aplicaciones y aislando a los usuarios sobre la interacción con distintas aplicaciones.

Luego de definir las principales herramientas SOA, vamos a analizar como desarrollar en sobre una arquitectura SOA y como nuestro desarrollo puede ayudar a la integración de las herramientas de Gobierno SOA.

# ¿Cómo se desarrolla sobre una arquitectura SOA?

El desarrollo sobre una arquitectura SOA debe siempre tener en cuenta las funcionalidades desarrolladas y debe consumir estos servicios y exponer las nuevas funcionalidades. Por lo tanto es importante tener un registro actualizado donde se registran los servicios desarrollados y si la empresa cuenta con sistemas heredados con servicios expuestos se debe realizar un relevamiento de los servicios desarrollados y reflejarlos en el registro. Que el registro esté actualizado y sea visual para todos los desarrolladores es vital para que no exista comportamiento replicado.

El desarrollador esta obligado a pensar los servicios de forma atómica y desacoplada. Un servicio nunca debe depender de una implementación sino de una interfaz que describe a dicha dependencia, agregando más complejidad a la tarea de diseñar pero disminuyendo el esfuerzo de mantenimiento ante un cambio. Promoviendo el desacoplamiento y la mantenibilidad.

El estilo arquitectónico SOA no es intrusivo, no nos obliga a utilizar una forma propia de SOA para desarrollo, ni un Paradigma de programación, ni un lenguaje, ni plataforma. Pero si nos obliga a no repetir funcionalidad publicada y publicar los servicios de tal forma sean útiles a otros sistemas. Por ejemplo podríamos programar con POO[15] y publicar la funcionalidad por medio de servicios o utilizar el patrón facade[SoaPatterns]. En este esquema los servicios van a seguir la arquitectura SOA, dando la libertad de diseñar nuestros objetos independientes de la arquitectura SOA. Lo más recomendado es utilizar una arquitectura en Capas[WikiCapas] sumado a la arquitectura SOA.

Una arquitectura en capas nos permite separar de forma lógica las incumbencias generales en capas. El objetivo de separar las incumbencias en capas, es el desacoplamiento. De esta forma podríamos cambiar ciertas capas si necesidad de modificar las demás.

Sumado a la arquitectura SOA con la arquitectura en Capas, aprovechamos lo mejor de los dos mundos.

[15] **POO:** Programación Orientada a Objeto

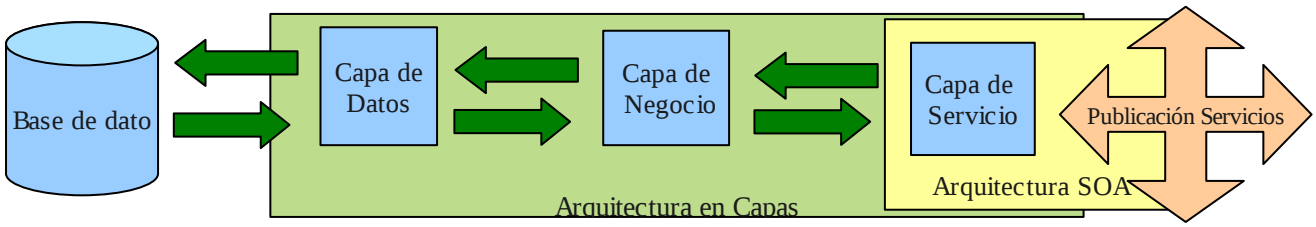


*Ilustración 5: Arquitectura en capas y SOA*

En la actualidad existen gran cantidad de framework que nos permiten desarrollar servicios web (CXF, Spring WS, Apache AXIS 1 y 2, Metro, etc.) y a la vez existen muchas formas de exponer estos servicios REST, SOAP, Mensajería, etc. y también existen diferentes formatos de datos JSON, SOAP, RSS, ATOM, etc. Todas estas tecnologías pueden conformar mis sistemas lo cual dificulta la comunicación ya que diferentes aplicaciones hablan diferentes idiomas.

Dada esta problemática surgió del mercado un estándar llamado Service Component Architecture (SCA), que propone una forma de desarrollo SOA. Basándose en el concepto de componentes promueve que un componente publica servicios que le permiten hablar idiomas diferentes (más usados) y entender la mayoría de los formatos de datos.

También a raíz de la diversidad de tecnologías para exponer servicios nace el ESB, que como dijimos anteriormente es un adaptador de diferentes “lenguajes”. Por lo tanto podríamos decidir utilizar un framework para exponer servicios y un ESB con su adaptador, o desarrollar con SCA. Dependiendo de los requerimientos se debe realizar una elección. Veamos en mayor detalle SCA para poder discutir qué tecnologías es mejor elegir.

# SCA

Podríamos concebir una aplicación como un conjunto de componentes de software interrelacionados. Todos estos componentes están construidos bajo las mismas o diferentes tecnologías. Estos componentes pueden correr sobre la misma máquina en un mismo sistema operativo y sobre una misma plataforma o en diferentes procesos, diferentes máquinas con diferentes plataformas y sistemas operativos. Sin embargo si una aplicación es organizada de esta manera se requiere una forma de crear los componentes y un mecanismo para describir cómo los componentes trabajan juntos. [IntroducingSCA]

Service Component Architecture (SCA) define un enfoque general para realizar estas dos cosas. SCA es un estándar de OASIS originalmente creado por diferentes vendedores como BEA, IBM, Oracle, SAP, etc. La especificación SCA define como crear un componente y como estos componentes interactúan para formar una aplicación. Los componentes en SCA pueden ser construidos en Java o en otros lenguajes y además permite interactuar con otras tecnologías como JEE, Spring o BPEL[16]. SCA define un mecanismo común de ensamblaje que indica como los componentes son combinados dentro de la aplicación.

SCA es una especificación; existen varias implementaciones entre las que podemos contar Apache Tuscany y Fabric3 [Fabric3].

## Componente

Una Aplicación puede ser construida con diferentes componentes. En una aplicación simple podríamos definir un componente como un simple objeto corriendo en un proceso. Y en una aplicación compleja podríamos definir un componente como un objeto que corre en varias máquinas utilizando diferentes mecanismos de comunicación.

Los componentes son como los átomos con los cuales una aplicación SCA es creada. Como átomos pueden comportarse de forma diferente y pueden ser ensamblados de forma diferente.

En el lenguaje SCA, un componente es una instancia de una implementación que debe ser apropiadamente configurada. La implementación es el código escrito en algún

[16] **BPEL:** Business Process Execution Language

lenguaje y la configuración esta escrita en un archivo en un lenguaje llamado SCDL[17] en dicho archivo se define los servicios que necesita el componente y los servicios que expone. En teoría un componente puede ser implementado en varias tecnologías y diferentes lenguajes.

Un componente contiene básicamente propiedades, servicios y referencias.

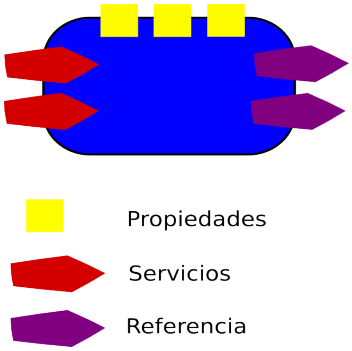


*Ilustración 6: Componente SCA*

Cada componente típicamente implementa lógica de negocio exponiendo dicha funcionalidad por medio de uno o varios servicios. Los servicios son descriptos de diferente manera dependiendo de la tecnología que fue implementando el componente por ejemplo si utilizamos java, el servicio será descripto por medio de una interfaz java.

Cada referencia que contiene un componente es una interfaz que contiene métodos que el puede invocar. Es decir un componente referencia una definición (una interfaz en java) y no a una implementación; esto permite mayor desacoplamiento y le da el control a SCA para que provea al componente las implantaciones que necesita.

Es común la idea de presentar lo que un objeto puede proporcionar por medio de una definición que describe la funcionalidad implementada en objetos. Esto trae muchas ventajas:

- Desacoplamiento: dado que los objetos no conocen la implementación, es más fácil reemplazar una implementación.

- Inyección de dependencias[Martinfowler]: permite darle la responsabilidad de creación de los objetos al contenedor de objetos en este caso SCA. Esto es sumamente útil ya que en el caso de SCA, el contenedor puede instanciar o hacer referencia a objetos que no se encuentra en el mismo equipo o si, sin que el programador necesite escribir código de comunicación. SCA inyecta las

[17] **SCDL:** Service Component Definition Language

dependencias de los objetos sin importar donde se encuentran y en que lenguaje fueron escrito.

Además de componentes y referencias, un componente puede definir una o más propiedades. Cada propiedad contiene un valor que puede ser leído del archivo de configuración SCDL cuando el componente es instanciado.

## Vinculaciones

Los servicios y las referencias permiten a un componente comunicarse, pero no describen como sucede esta comunicación, este es el trabajo de las vinculaciones (en Inglés Bindings).

Una aplicación se comunica con diferentes aplicaciones por diferentes protocolos y diferentes modelos de comunicación.

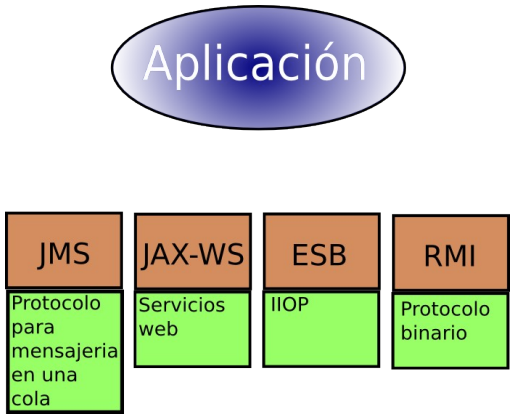


*Ilustración 7: Ejemplo de comunicación de una aplicación*

SCA encapsula la comunicación con diferentes protocolos por medio de las vinculaciones. De esta forma el desarrollador no necesita aprender diferentes protocolos y Apis. Existe una vinculación que puede resolver la comunicación con una tecnología. De este modo existe una vinculación para Web services soap, vinculación para JMS[18], para comunicación con EJB[19], etc.

[18] **JMS:** La API Java Message Service (en español *servicio de mensajes Java*), también conocida por sus siglas JMS, es la solución creada por Sun Microsystems para el uso de colas de mensajes. Este es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma Java 2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona.

[19] **ESB:** Los Enterprise JavaBeans son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE (ahora JEE 6.0) de Oracle Corporation (inicialmente desarrollado por Sun Microsystems). Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor que son, precisamente, los EJB.

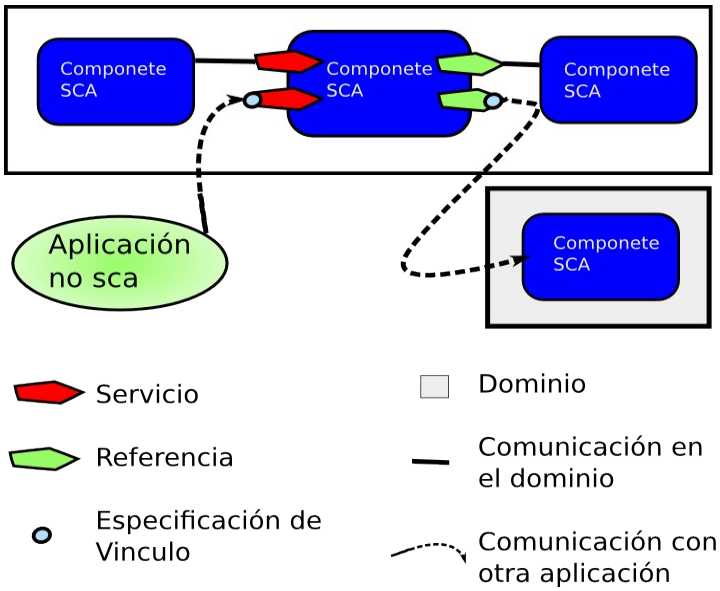


*Ilustración 8: Ejemplo de vinculación*

Una vinculación especifica como debe ser la comunicación entre un componente SCA y algo más. Dependiendo de la comunicación se debe o no especificar la vinculación, en el caso en que componentes del mismo dominio se comuniquen no es necesario especificar una vinculación; el runtime determina que vinculación utilizar.

En el caso de la comunicación sea con un componente SCA de otro dominio u otra aplicación no SCA es necesario definir una o más a vinculaciones para esta comunicación. Cada vinculación define un particular protocolo que puede ser usado en la comunicación con esta referencia o servicio. Una referencia o servicio puede tener múltiples vinculaciones permitiendo comunicarse con diferentes software remoto por diferentes caminos como podemos ver en la Ilustración 8.

La ventaja que conlleva separar la lógica del negocio plasmada en el componente de su forma de comunicarse (vinculaciones) son mayor flexibilidad y facilidad de diseño y desarrollo.

## La composición

Si el componente es un átomo para SCA, la composición es una molécula. Una composición es un grupo de componentes combinado de forma útil, y además puede ser combinado de diferentes formas.

La composición es la base de la promesa más apasionante de una SOA: la agilidad. Una vez que se cuenta con un buen número de servicios, la creación de nuevas funcionalidades útiles para el negocio es sólo cuestión de conectar los servicios adecuados.

En la práctica, la composición requiere contar con el conjunto adecuado de servicios, y que definan el conjunto de operaciones adecuadas. Además, la tecnología empleada para componer funcionalidades con estos servicios debe estar impulsada desde una perspectiva puramente de negocio para permitir hacer cambios rápidos en las aplicaciones.

Un compuesto podríamos definirlo como una construcción lógica que genera funcionalidad a partir de servicios expuestos por otros componentes y a la vez es capaz de exponer esta nueva funcionalidad como servicio.

Podemos ver un compuesto como lo muestra el siguiente ejemplo:

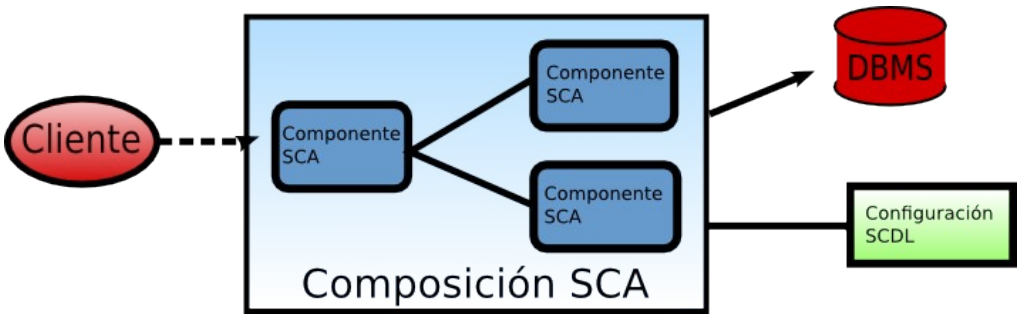


*Ilustración 9: Ejemplo de uso de composición en SCA*

Los componentes que forman un compuesto se pueden ejecutar en un solo proceso en un único equipo o distribuidos a través de múltiples procesos en varios equipos. Una aplicación completa podría ser construida a partir de un solo compuesto, como en el ejemplo que se muestra, o combinar varios compuestos. Los componentes que conforman un compuesto podrían utilizar la misma tecnología, o puede que sea construida utilizando diferentes tecnologías de diferentes plataformas.

## Cables y promoción.

Un cable es una abstracción que representa una relación entre una referencia y un servicio que satisface las necesidades de esa referencia. Una referencia de un componente es conectada a un servicio en otro componente usando un cable.

Al igual que los componentes exponen servicios, una composición puede exponer uno o más servicios. Los servicios pueden ser implementados por componentes dentro de

la composición. La composición puede exponer o **promocionar** este servicio el cual es expuesto por el componente dentro de la composición.

## Dominios

Los Dominios son un concepto importante en SCA. Dado que SCA define un sistema distribuido pero no define como los componentes interactúan entre si.

Un dominio puede contener compuestos que pueden contener componentes los cuales puedes ser ejecutados en uno o más procesos en una o más computadoras. Como podemos ver en la Ilustración 10

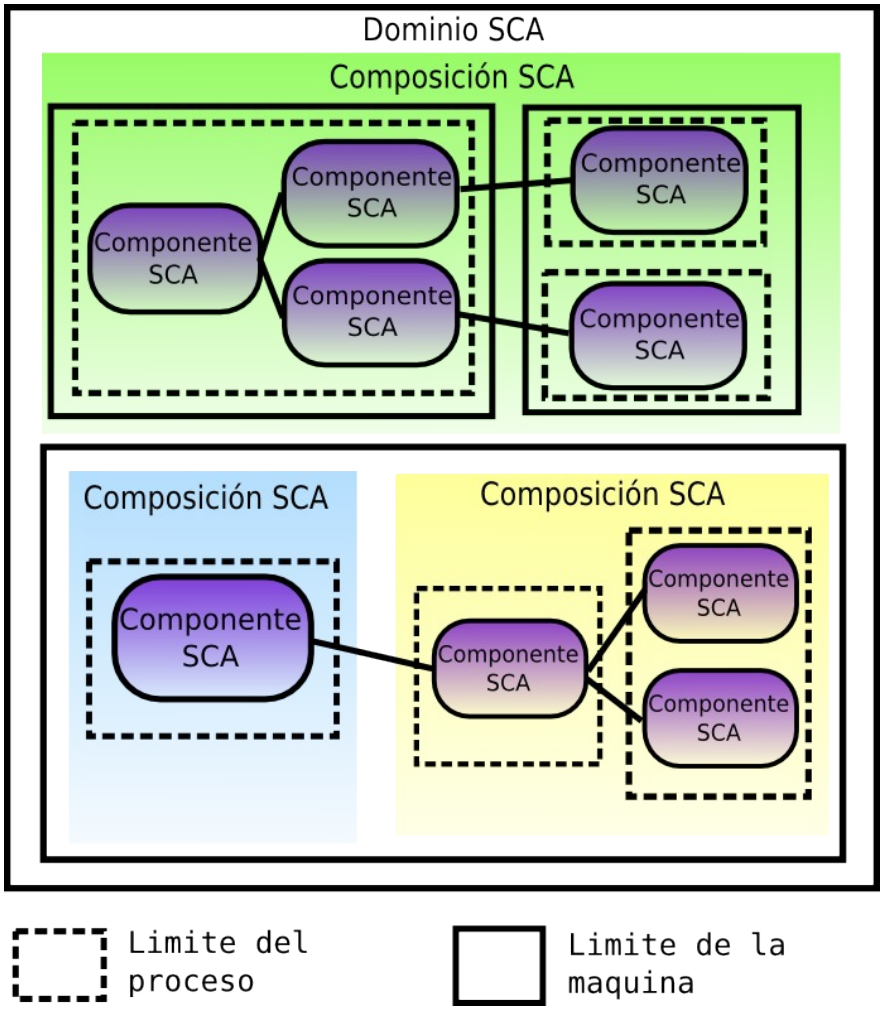


*Ilustración 10: Ejemplo de Dominio SCA*

SCA no define cómo los componentes en las aplicaciones interactúan con diferentes dominios. SCA define el dominio de la aplicación donde los componentes y composiciones se realizan e interactúan. Esto permite que el contenedor de componentes y composiciones pueda hacer optimizaciones útiles. La vida de un desarrollador de SCA es mucho más sencilla dentro de un dominio, por ejemplo, se pueden evitar las

complejidades inherentes a la configuración de aplicaciones de múltiples proveedores o protocolos, descansando en el contenedor SCA. [OracleSCA]

Cuando queremos comunicar diferentes dominios de diferentes proveedores lo podemos hacer con protocolos estándar como por ejemplo web services SOAP. Esto lo podemos realizar con vinculaciones. Las vinculaciones nos permiten hablar con diferentes dominios SCA o con otras aplicaciones no SCA. Se puede ver un ejemplo gráfico en la Ilustración 11

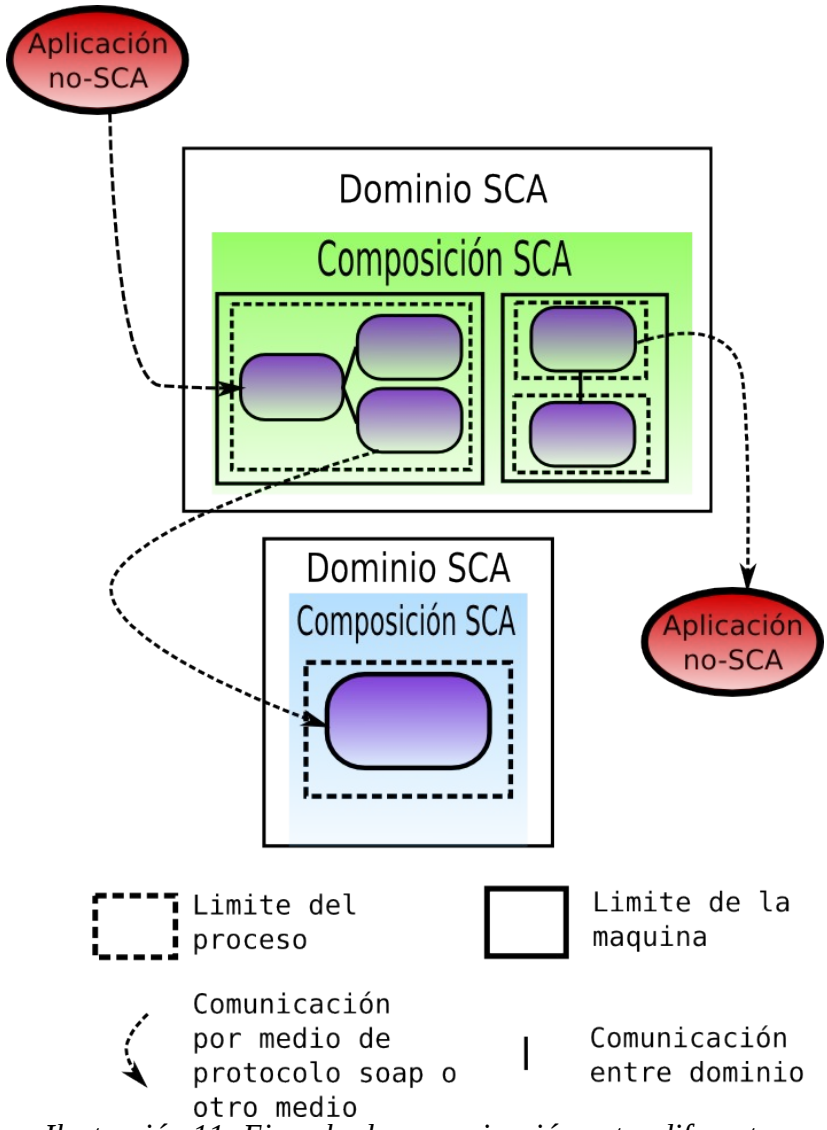


*Ilustración 11: Ejemplo de comunicación entre diferentes dominios SCA y aplicaciones no SCA*

Un Componente además de proporcionar servicios a sus propios clientes, también puede proveer sus servicios a otros componentes de su dominio y a clientes de otros dominios.

# Recapitulación del marco teórico

En el desarrollo teórico explicamos conceptualmente SOA, los servicios y el gobierno SOA. Se describieron las herramientas SOA más utilizadas. Para luego centrarnos en el desarrollo SOA, con SCA. En el desarrollo de los temas se puede notar que hay funcionalidad replicada en las herramientas del Gobierno SOA y en SCA. Es necesario recordar que para diseñar SOA no es obligatorio tener todas las herramientas descriptas anteriormente, solo se deben utilizar las que se necesitan. Por ejemplo el registro es necesario siempre, dado que es imprescindible para no tener comportamiento replicado. Pero si se desarrolla con SCA, no es necesario utilizar la funcionalidad de adaptador de un ESB.

Por estas razones es importante que el arquitecto SOA, analice los requerimientos y utilice las herramientas justas para sus necesidades. No es bueno utilizar herramientas solo porque existen. El objetivo es el beneficio y valor agregado que pueden dar estas herramientas.

Luego de haber visto las principales herramientas SOA y la forma en la cual podemos desarrollar nuestra SOA, nos centraremos en un caso práctico el cual nos permitirá explicar que se puede desarrollar Web 2.0 con herramientas Open Sources. Además analizaremos las ventajas y desventajas que esto trae el desarrollo Web 2.0 con SOA.

Elegiremos qué herramienta de gobierno SOA necesitamos para desarrollar y qué modelo de desarrollo nos conviene utilizar. Además buscaremos en el mercado cual es la herramienta Open Source más utilizada y conveniente para nuestras necesidades.

# Caso de estudio : blog[20]

Esta Tesis trae como objetivo, implementar la Arquitecturas de Software Orientadas a Servicios en la Web 2.0, basándose en herramientas Open Source. Para esto nos valdremos de un ejemplo práctico el cual se encuentra en http://code.google.com/p/nornas/ y se puede acceder a su código por cualquier cliente svn[21] o bajar el código compilado desde el apartado download. Un Weblog es una aplicación simple que refleja el poder de la web 2.0. La comunidad publica (postea) conocimiento, pensamientos, fotos, historias, etc. y los seguidores pueden leer las publicaciones y comentarlas.

Los blogs son un ejemplo claro de la web 2.0 dado que:

- Permiten sociabilizar y intercambiar conocimiento.
- Dan facilidad de navegación con funcionalidad ajax.
- Permiten comunicarse con diferentes sitios webs.

Por lo tanto tomaremos como ejemplo el desarrollo de un motor de blogs simple y probaremos que se puede desarrollar con tecnología SOA Open Sources y se analizaran ventajas y desventajas.

Cuando se analiza la aplicación de un estilo arquitectónico es esencial que nos centremos en los requerimientos no funcionales, sin olvidar los funcionales.

Los requerimientos no funcionales especifican propiedades del sistema como restricciones de ambiente y desarrollo, performance, dependencias de plataformas, mantenibilidad y confiabilidad. Estos requisitos, en lugar de definir lo que la aplicación hace, definen cómo la aplicación proporciona las funcionalidades requeridas. Además son el velocímetro para medir la aplicación de un estilo arquitectónico.

Los requerimientos funcionales por lo general son discretos en el sentido de que agregando o modificando algunas líneas de código en unos pocos lugares es suficiente para implementarlos, mientras que los requerimientos no funcionales son, por lo común, transversales en el sentido de que es necesario agregar o modificar código en todas

[20] **Blog:** también una bitácora, es un sitio web periódicamente actualizado que recopila cronológicamente textos o artículos de uno o varios autores, apareciendo primero el más reciente, donde el autor conserva siempre la libertad de dejar publicado lo que crea pertinente.
[21] **SVN:** Subversion es un sistema de control de versiones diseñado específicamente para reemplazar al popular CVS. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como svn por ser el nombre de la herramienta utilizada en la línea de órdenes.

partes para implementarlos. En consecuencia no prever un requerimiento no funcional suele ser mucho más costoso que no tener en cuenta un requisito funcional.

# ¿Por que elegir SOA para el desarrollo Web 2.0?

Las ventajas de un desarrollo SOA son los mismos para una aplicación Empresarial como para una aplicación Web 2.0. Pero los requerimientos no funcionales de estas aplicaciones son diferentes.

Veamos los requerimientos no funcionales de una Web 2.0 :

- Capacidad de escalar: Una características de las Web 2.0 es su rápido crecimiento, por lo tanto es imprescindible que pueda rápidamente escalar a varios servidores de una forma distribuida.

- Performance: una aplicación web 2.0 debe ser rápida, nunca debe dar sensación de pesadez o lentitud. Esto se logra mediante técnicas de ajax que permiten que la pagina se cargue por sectores mostrando al usuario sectores más livianos y luego los más pesados.

- Mantenibilidad: si bien todas las aplicaciones deben ser mantenibles, las aplicaciones Web 2.0 están en contante evolución incorporando cada vez más funcionalidad por lo tanto es importante que el código sea claro, simple, fácil de mantener y no debe haber código duplicado.

- Facilidad de uso: la usabilidad es un tema prioritario en estas paginas, dado que contaremos con todo tipos de usuarios.

- Soporte los navegadores más usados: las paginas web 2.0 deben tener en cuenta los navegadores más utilizados para poder usado por diferentes usuarios que usan diferentes sistemas operativos y dispositivos.

- Comunicación con otras paginas: las paginas web 2.0 buscan ampliar su mercado y estar en todas partes, por ejemplo publicando un post en Bloogler se ve reflejado en Google Buzz.

Estos son los los requerimientos genéricos de una web 2.0 ahora veamos como SOA nos ayuda con ellos:

- Capacidad de escalar: La arquitectura SOA es una arquitectura distribuida lo que nos permite tener nuestra aplicación corriendo en diferentes ordenadores con diferentes plataformas.

- Performance: En este requerimiento SOA no nos ayuda, dado que al agregar más capas desde la base de datos al navegador el proceso de transporte de datos es más lento. La aplicación debe serializar datos en SOAP por ejemplo y el cliente luego debe desserializarlos, para poder utilizarlos. Pero al permitir correr diferentes servicios en diferentes maquinas, es posible disminuir la lentitud agregando más servidores y cargando la web con técnicas de ajax.

- Mantenibilidad: SOA nos ayuda a desarrollar código más claro y reutilizable. SCA en especial es una especificación que promueve las mejores prácticas de desarrollo. Si desarrollamos siguiendo SCA obtendremos alta reutilización de código distribuido.

- Comunicación con otras paginas: SOA promueve la comunicación y facilita la misma. Utilizando SCA por ejemplo podríamos utilizar servicios web o ser referenciados en diferentes protocolos y métodos de comunicación sin tener que escribir una sola linea de código para realizar esta comunicación.

Para los requerimientos “Facilidad de uso” y “Soporte de los navegadores más usados” el uso de SOA no aporta nada, es indistinto si se usa o no SOA ya que son requerimientos solo de la capa de presentación de la aplicación.

En conclusión utilizar una arquitectura SOA nos proporciona mayores ventajas en el momento de desarrollar aplicaciones web 2.0, sin embargo recordemos que como desventaja se veía disminuida la performance. Esto no es un dato menor dado que es importante para la web 2.0 la performance y la agilidad del sitio. El modelo de desarrollo SOA, promueve la reutilización de servicios y la separación en capas, agregando complejidad que afecta la performance.

Gracias a técnicas de Ajax y la capacidad de crecer verticalmente, la performance no debería disminuir de forma significativa. Las técnicas de ajax del lado cliente darán al usuario una sensación de agilidad y recargaran la pagina de forma progresiva. La capacitad de crecer verticalmente que nos brinda la arquitectura SOA, nos permite agregar servidores para que se procese parte de nuestra aplicación mucho más rápido.

# ¿Porque optar por soluciones Open Sources?

Como principal ventaja del software Open Sources se pensaría en el bajo costo dado que se relaciona Open Sources como gratis y esto no es así. Es un error pensar en

el Open Sources como una solución barata dado que el software puede tener costo (en nuestro caso no es así, todos los frameworks son open sources y gratuitos) , pero además el software tanto Open Sources como propietario tiene costos ocultos. Como costo oculto podríamos nombrar la falta de soporte, o falta de documentación. Estos costos ocultos hay que tenerlos en cuenta a la hora de optar por un software.

El uso de frameworks open sources para el desarrollo de aplicaciones trae aparejado muchos beneficios que dependen de la comunidad que respalde el framework; cuanto mayor es la comunidad mayor va a ser el soporte, la documentación, calidad y los usuarios que usan el framework:

- Soporte: El soporte brindado puede ser mayor a una aplicación propietaria y la información suministrada por foros de problemas es basta.

- Documentación: Si bien no es común que un software open source este bien documentado, la información suministrada por blogs, foros, libros es mayor en muchos casos que la de productos comerciales.

A la vez existen beneficios intrínsecos del software open source:

- Libertad de uso: podemos usar el programa sin ninguna restricción y además podemos modificar el software y poder comercializar estas mejoras siempre y cuando brindemos usemos la misma licencia.

- Libertad de aprender: podemos aprender del código fuente, aprender como fue hecho el framework y saber como realmente funciona. De esta forma el grupo de trabajo no esta limitada a ver solo la documentación sino que puede debugear el código fuente y palpar lo que hace realmente.

- Libertad de modificar: Si existe un bug, es posible corregirlo y publicar su corrección para que se encuentre en futuras versiones. Además podemos extender el framework como queramos, desarrollando nueva funcionalidad que luego podemos publicar para que la comunidad la pruebe y use.

Estas ventajas no son menores, dado que el conocimiento es el arma más importante en un grupo de desarrollo informático y el open source permite cultivar el conocimiento, dando nos el privilegio de leer las lineas de código y ver como funciona.

Apostar a el software open source es apostar al capital humano, con un grupo consolidado puede aprender y desarrollar mejor a partir del código fuente. Los desarrolladores aprenden como funciona el framework viéndolo funcionar.

Como desventaja se ve que al estar el código disponible, puede haber mayores problemas de seguridad, ya que cualquier programador con malas intenciones puede buscar fácilmente bugs o debilidades que pueda explotar para su propio beneficio.

Estos beneficios son propios del Open Source, y además existen razones para pensar en desarrollar SOA Open Sources:

- Amplia variedad de herramientas en el mercado: La variedad de propuesta para herramientas SOA es grande y en algunos casos mayor al abanico de opciones de software comercial.
- Amplio uso y aceptación: Las herramientas open source SOA son usadas por importantes empresas y con una gran comunidad.

Dado estos beneficios es muy importante utilizar herramientas open source en desarrollos SOA. Utilizar estas herramientas no solo beneficia a quien lo usa sino a toda una comunidad.

# ¿Que plataforma utilizar?

Si decidimos utilizar tecnología Open Source para el desarrollo, la opción más clara es utilizar la plataforma Java ya que cuenta con la más amplio abanico de herramientas SOA. La comunidad Java fue vanguardista con respecto a SOA. En java están escritos la mayor cantidad de framework SOA open source y los más utilizados.

Otra opción es utilizar la plataforma .NET la cual provee herramientas SOA, las cuales en su gran mayoría son propietarias. Además la comunidad open sources de .NET es mucho menor a la de Java. Por esta razón descartamos .NET.

Otras plataformas open sources pueden ser interesantes para desarrollar, PHP, Ruby o Python, son muy relevantes para el mundo web, pero no para el mundo SOA. Si bien existe soporte y herramientas SOA en estas plataformas, es mucho menor que el del mundo Java.

La plataforma Java fue acompañada por grandes proveedores de software como IBM, BEA, Oracle, Sun (en su momento) para convertirse en la plataforma ideal para desarrollar SOA. Dados la condición de Java en el mercado es muy conveniente utilizarlo para nuestro desarrollo.

# ¿Que herramientas necesitamos en nuestro desarrollo?

Es importante recordar que para desarrollar una arquitectura SOA, no es necesario utilizar todas las herramientas y frameworks que hay en el mercado. Una tarea del arquitecto SOA es revisar los requerimientos y analizar que herramientas son necesarias.

Para el desarrollo del motor de blog SOA, va a ser necesario registrar los servicios por lo tanto se debe utilizar un software de registro de servicios. Al registrar los servicios, se crea un repositorio de servicios que sirve para que no se dupliquen y como documentación de los mismos.

Existen diferentes herramientas y framework SOA, y entre ellos hay funcionalidad que se solapa. Y a la vez existen diferentes caminos de desarrollo por lo tanto, una decisión que hay que tomar antes de desarrollar, es qué modelo de desarrollo utilizar. Una opción es desarrollar nuestra aplicación utilizando un framework de web services para exponer servicios soap y nuestra aplicación se comunicara con el mundo por medio de un ESB que se encargara de traducir los mensajes a soap.

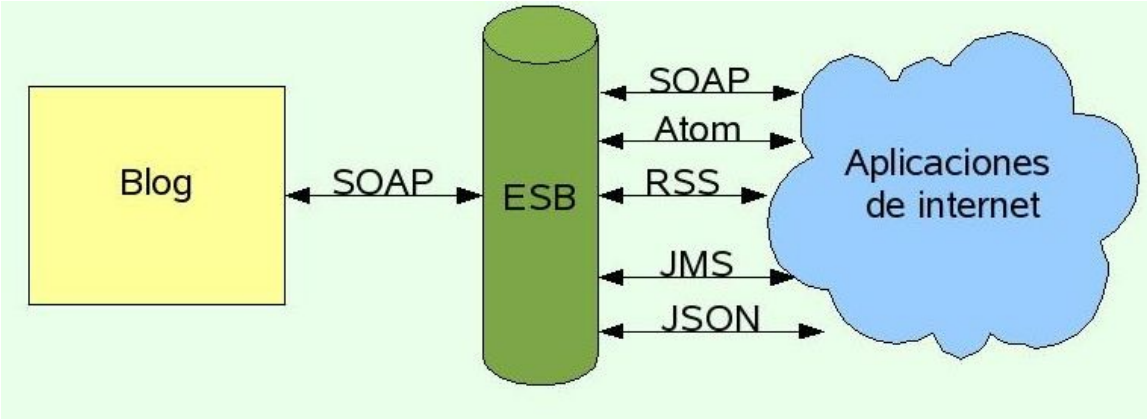


*Ilustración 12: Arquitectura usando ESB*

Este enfoque nos obliga a utilizar un ESB para relacionarnos con otras aplicaciones de la web, dado que las demás aplicaciones se comunican y exponen sus servicios de diferente manera.

Otro camino posible es desarrollar según el estándar SCA. Usando SCA no necesitamos un ESB para comunicarnos con aplicaciones web ya que SCA prevé esto.

Por medio de los vínculos de SCA podemos comunicarnos con diferentes aplicaciones con diferentes protocolos.

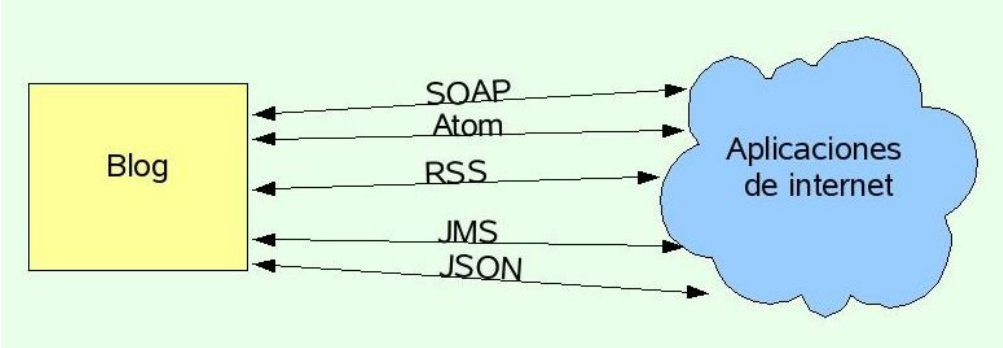


*Ilustración 13: Arquitectura usando SCA*

Podríamos usar un ESB para dos propósitos, unificar el canal de comunicación de todas las aplicaciones de mi empresa y permitir que aplicaciones dialoguen sin importar el protocolo de comunicación que usen. Como el motor de blog es una única aplicación que se podría publicar en internet o en una intranet, no necesita tener un canal de comunicación unificado. Y si usamos SCA no vamos a necesitar que un ESB haga adaptaciones de protocolo o lenguaje ya que SCA provee esta funcionalidad.

Usar SCA sin un ESB es la mejor opción para el desarrollo del motor del blog. SCA nos provee la posibilidad de comunicarnos con cualquier aplicación sin importar la técnica de comunicación que use y además nos provee un modelo de desarrollo facilitando la mantenibilidad, agilidad de desarrollo y un modelo de desarrollo distribuido.

Una funcionalidad interesante a introducir en el blog es poder marcar workflows configurables en diferentes situaciones por ejemplo que cuando publicamos un post, que se publique el post y avise a todos los seguidores del blog por mail o al comentar un post, se guarde el comentario y se envié por mail a el dueño del blog una notificación. Para realizar esta funcionalidad y que sea realmente configurable es conveniente usar un motor BPEL.

En resumen:

- Para el desarrollo es más conveniente el uso de la especificación SCA.
- Para el Gobierno SOA solo necesitamos una herramienta de registro y un motor BPEL, no es necesario ni un ESB, ni ESP y CEP (dado que no tenemos procesamiento complejo de eventos) , ni EIA (dado que no contamos con aplicaciones heredadas o legacy).

A continuación se analiza que herramientas open source que nos provee el mercado para desarrollar el motor de blog.

## ¿Que implementación SCA utilizar?

Existen actualmente varias implementaciones open source de la especificación SCA de la cuales se destaca: Apache Tuscany y Fabric3. Apache Tuscany es implementación de referencia de SCA. Su comunidad es mayor a la de Fabri3. Los sitios web de Tuscany y OASIS han recogido de forma colaborativa documentación extensa. Los documentos de la especificación de SCA y su tecnología relacionada están bien documentadas y son perfectamente entendibles.

Apache Tuscany dio a luz su primera versión beta en el año 2006, su versión a en este momento es 1.6 (versión publicada en febrero del 2010). Tuscany presenta un desarrollo sostenido y rápida corrección de errores.

Considerando los puntos: comunidad, documentación y estabilidad, podemos afirmar que Apache Tuscany es la mejor opción como solución SCA.

## ¿Que herramienta de registro utilizar?

El objetivo de tener un registro de las funcionalidades desarrolladas es que todos los desarrolladores conozcan las funcionalidades y no exista duplicidad de servicios. Para esto se podría utilizar diferentes herramientas de propósito más general, como una wiki o una base LDAP, pero es mejor utilizar una herramienta de propósito particular. En el mercado existen herramientas open source para el registro de servicios. Entre los más importantes podemos mencionar a MuleSource Galaxy y Registro de WSO2.

**MuleSource Galaxy**: Esta basado en un diseñador de repositorios para la gestión de artefactos de software basados en SOA. Entre ellos se incluye la configuración del ESB Mule, WSDL, archivos xml y configuración de Spring. Es una perfecta herramienta de registro cuando se utiliza ESB mule ya que se integra totalmente con ese producto.

**Registro de WSO2**: Esta diseñado para almacenar, catalogar, indexar y gestionar metadatos de empresa relacionados con artefactos SOA. Incluye el control de versiones y su rapidez lo hace un buen candidato para sistemas empotrados.

El producto Galaxy tiene soporte para las mismas funcionalidades generales que el registro de WSO2, como puede ser la categorización de recursos, la monitorización y la gestión de ciclo de vida y de la dependencia. Además, su versión 1.5 incluye nuevas funcionalidades como la replicación (disponible solo la versión de pago) tiene soporte para scritps y una API de eventos.

El producto de registro WSO2 es muy sencillo de utilizar y en este punto supera a Galaxy gracias a su mejor infraestructura para Atom/Rss y sus funcionalidades de control de versiones y de restauración de versiones anteriores.

Las funcionalidades más atractivas de Galaxy se encuentran en la versión de pago mientras que la herramienta de registro de WSO2 es 100% libre y gratuita. Por esta razón y la facilidad de uso, optamos por la herramienta de registro de WSO2.

# ¿Que herramienta BPM utilizar?

Como dijimos anteriormente el blog va tener funcionalidad de workflow, permitiendo que el blog se adapte a diferentes realidades como por ejemplo: uso empresarial en una intranet o un blog personal en la web. Utilizar BPEL nos permitirá hacer una aplicación más genérica y configurable.

En el mercado existen gran variedad de herramientas BPM y BPEL open source, las más importantes son:

- **Intalio BPM (edición community)** : Herramienta BPM rica en funcionalidades que utiliza la noción de modelado de procesos de negocio BPMN[22] para generar orquestaciones basadas en BPEL. Al ser una versión open source de un producto comercial utiliza librerías de la versión de pago, este software fue concebido para permitir el desarrollo y luego cuando se ponga en producción se compre la versión estándar.

- **Motor ActiveBPEL**: Motor BPEL eficiente y sumamente cuidado. Los modelos pueden diseñarse utilizando su herramienta Designer que es gratuita pero no es open source. La funcionalidad más importante, como puede ser las

[22] **BPMN:** Business Porcess Modeling Notion (en español Notación para el Modelado de Procesos de Negocio) es una notación gráfica estandarizada que permite el modelado de procesos de negocio, en un formato de flujo de trabajo (workflow). BPMN fue inicialmente desarrollada por la organización Business Process Management Initiative (BPMI), y es actualmente mantenida por el OMG (Object Management Group), luego de la fusión de las dos organizaciones en el año 2005.

instancias de procesos persistentes a una base de datos o el control de versión de los procesos, únicamente esta disponible en la versión Enterprise.

- **Apache ODE**: Apache ODE (Orchestation Director Engine) es un motor BPEL de ejecución de procesos. Su API permite extenderlo de muchas maneras y por lo tanto, no esta limitado a utilizar únicamente SOAP. Es un motor liviano que puede integrarse fácilmente con productos de Apache como Service Mix. Apache ODE no posee editor gráfico para deseñar los procesos pero se puede contar con un plugin que se puede instalar en Eclipse[23].

- **Jboss JBPM**: Es un motor de workflow maduro, eficiente y ligero que va siempre de la mano de la herramienta de modelado basado en Eclipse. Usa su propio lenguaje de grafo en xml llamado jPDL (jBPM process Definition Language) y tiene soporte para todos los nodos de modelado principales, como puede ser las decisiones y las bifurcaciones. Se puede extender de manera sencilla y no está restringido su uso a ningún entorno de despliegue. A diferencia de otras alternativas, no existe una actualización a la versión comercial y no hay ninguna funcionalidad que esté restringida.

- **ObjectWeb Bonita**: Se trata de un motor de workflow potente y compatible con XPDL. Es un proyecto maduro y bien documentado. Incluye excelente integración con herramientas gráficas de actividades humanas (por ejemplo, un generador de formularios) No dispone de ningún editor de software libre y necesita el servidor de aplicaciones JOnAS (Java Open Application Server)

- **WSO2 Bussiness Process Server**: El producto está basado en Apache ODE e incluye una interfaz administrativa basada en la Web y funcionalidades de simulación.

Si buscamos una solución 100% open source y sin restricciones se debe elegir Apache ODE, Jboss JBPM, Bonita o WSO2 Bussiness Process Server. Una restricción importante es que el software que deseamos construir no dependa de un sistema operativo, base de datos o servidor. Bonita depende del servidor JOnAS por lo tanto lo eliminamos como opción. Si analizamos la comunidad y la documentación para seleccionar entre WSO2 Bussiness Process Server, Apache ODE y JBPM, el ganador sería JBPM dado que fue uno de los primeros productos BPM open source, su comunidad es bastante grande y activa y la documentación es basta.

[23] **Eclipse:** es un entorno de desarrollo integrado de código abierto multiplataforma. Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Jboss JBPM es el producto más maduro y probado en el mercado pero es un producto bpm, que como explicamos anteriormente BPM puede ser usado para la orquestación de servicios web, pero no fue concedido para esto y por lo tanto se debe desarrollar modificaciones para cumplir la funcionalidad de orquestación. En el caso de JBPM, es necesario desarrollar un mecanismo para publicar su funcionalidad como web services. Dado que se quiere minimizar los costos de desarrollo se opta por no elegir JBPM.

WSO2 Bussiness Process Server es un motor BPEL de WSO2, empresa centrada en el desarrollo de open source de soluciones SOA. WSO2 Bussiness Process Server es un desarrollo que toma a Apache ODE de base y le agrega herramientas administrativas las cuales facilitan el desarrollo y la administración del motor BPEL

Contamos con todas las herramientas Open Source para comenzar a desarrollar, en la secciones siguientes especificaremos la arquitectura para luego señalar las herramientas de desarrollo.

# Desarrollo de arquitectura del Blog

Un requerimiento no funcional importante para motor de blog es que sea capaz de ser usado en diferentes plataformas. El motor de blog debe poder utilizar base de datos de diferentes proveedores y ser lo más flexible posible. De esta forma se podría utilizar en una intranet de una empresa como sistema de blog y comunicación con sus empleados o como un blog común expuesto en internet.

Para satisfacer este requerimiento es necesario pensar en una arquitectura modular y por capas donde cada capa se abstraiga y cumpla una responsabilidad propia. La capa de servicios va a ser contenida por un dominio SCA y el cliente podrá ser un cliente que utilice SCA o cualquier otro framework cliente de servicios web.

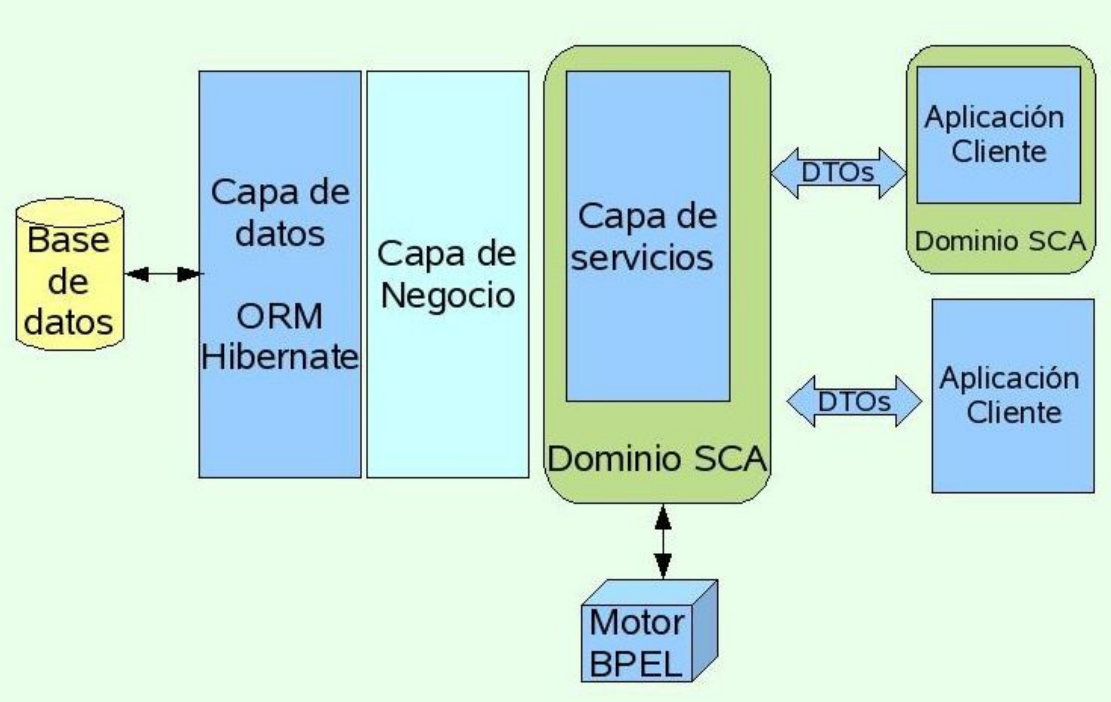


*Ilustración 14: Arquitectura del Blog*

Es necesario separar la aplicación en dos, una parte servidor y otra cliente de esta forma la aplicación servidor expone los servicios sin importar el consumidor y una aplicación cliente utiliza esos servicios para mostrarlos de forma amigable al usuario final. Se podría utilizar cualquier protocolo para comunicar el servidor con el cliente dado, que sera indistinto a la hora de desarrollar por que se usará un framework SCA. SCA permite un solo servicio con diferentes protocolos de comunicación y end points. Dado que el cliente va a consumir un solo protocolo vamos a optar por publicar servicios basados en SOAP.

Al separar la aplicación cliente interfaz de la api servidor podemos tener muchos clientes; se puede desarrollar un cliente de escritorio, un cliente móvil o una pagina web, que permitan realizar todas las acciones propias de un blog.

Para que se transmitan objetos livianos es necesario utilizar DTOs (data transfers Objects). Los DTOs son objetos que solo representan los datos que son necesarios mostrar. Esto permite que solo se envién al cliente los datos que necesita y además, no es necesario enviar los  objetos del modelo, que no es recomendado ya que los objetos del modelo tienen lógica del negocio que no es bueno hacer pública y a la vez son más pesados. Los DTOs se van a serializar y van a ser enviados al cliente por medio del protocolo SOAP. Con la separación de los objetos del modelo con los DTOs, se obtiene como ganancia:

- Los DTO utilizan anotaciones para ser serializados, por lo tanto el modelo negocio es independiente de estas anotaciones y de cualquier tecnología, lo que nos permite reutilizarlo o migrar tecnología sin tocar los objetos del modelo de negocio.

- Como dijimos anteriormente los objetos DTOs muestran solo lo que los servicios deben publicar.

- Los DTOs no tienen lógica ni métodos por lo tanto son más livianos.

La capa de servicio va a estar dentro de nuestro dominio SCA y la aplicación cliente en otro, lo que permite que estas aplicaciones estén desacopladas. La capa de servicio va a utilizar un motor BPEL como ya lo dijimos anteriormente. Para ello va a utilizar la funcionalidad de integración con BPEL de Tuscany.

La capa de datos es la encargada de buscar datos a una base de datos, pero no debe depender de un motor de base de datos de algún proveedor por lo tanto es necesario utilizar algún framework que nos permita hacer esto. Frameworks ORM[24] existen muchos en el mercado, pero sin duda el más popular es Hibernate; su reinado en el mundo de los framework ORM es innegable. Con Hibernate podremos hacer la aplicación independiente del motor de base de datos que utilice.

Vimos la arquitectura a alto nivel, ahora veremos como se compone físicamente la aplicación, con un diagrama de paquetes:

[24] El **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos, aunque algunos programadores prefieren crear sus propias herramientas ORM.

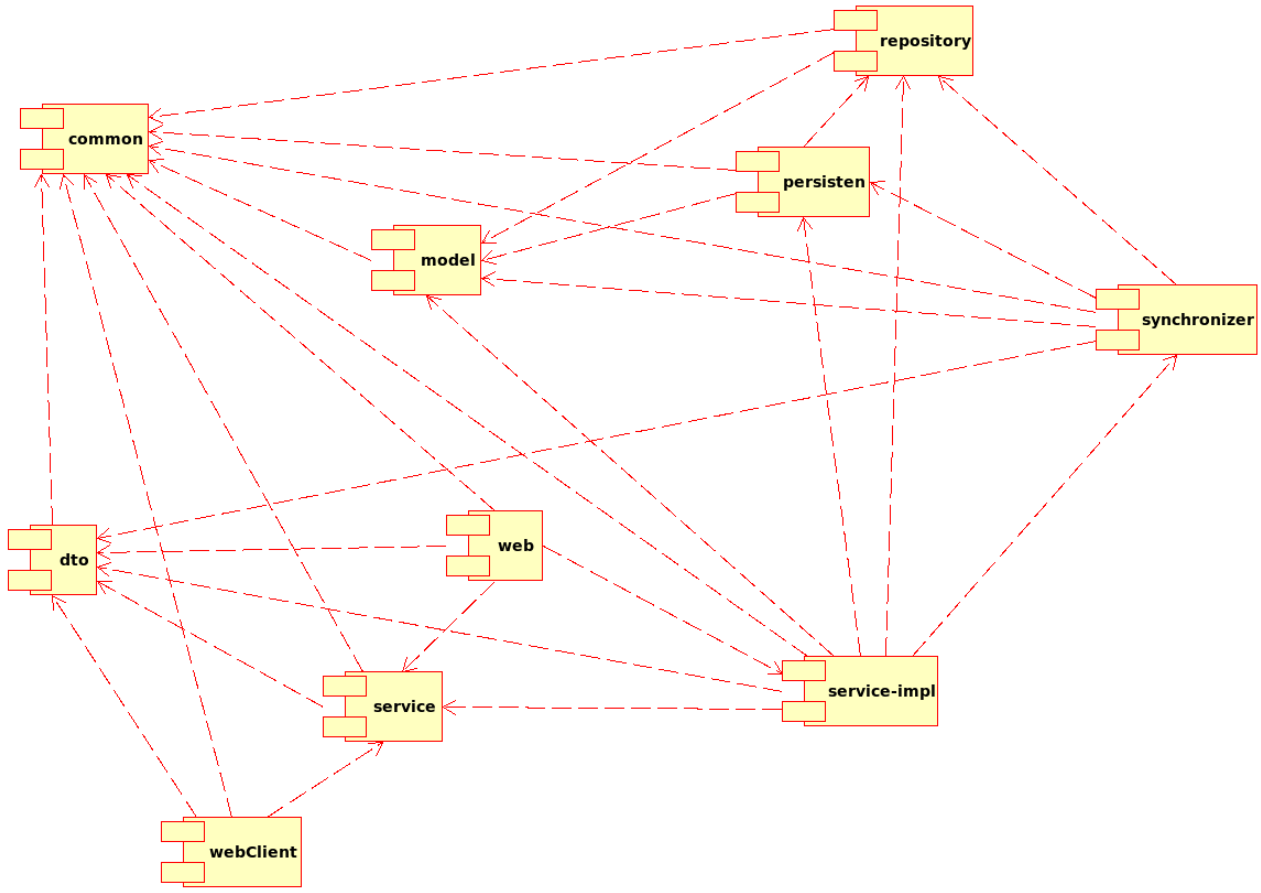


*Ilustración 15: Diagrama de paquetes*

La división de paquetes nos da mayor flexiblilidad y mantenibilidad, dado que tiene como objetivo que se pueda cambiar algún paquete sin cambiar los demás.

Hay paquetes que contienen declaraciones y otros la implementación de funcionalidad, esto ayuda al desacoplamiento. De esta forma podemos cambiar el paquete de implementación sin modificar los otros paquetes siempre y cuando no cambie una declaración.

Los paquetes son:

**Common**: Contiene clases, declaración de excepciones y declaraciones de tipos comunes a todos los paquetes. Todos los paquetes dependen de este y common no depende de ningún paquete.

**Model**: Contiene la representación lógica del modelo de negocio, modelada en clases. Solo depende de common.

**Repository**: Contiene la declaración de los objetos de acceso a datos. Depende de common y model.

**Persistent**: Contiene la implementación de los objetos de acceso a datos. Depende de common, model y repository. Este paquete contendrá todos los objetos que permiten persistir y recuperar de la base de datos a los objetos del modelo para ello utilizará Hibernate.

**Dto**: Contiene la implementación de los objetos transferencia de datos. Solo depende de common.

**Service**: Contiene la declaración de los servicios. Depende de common y dto.

**Synchronizer**: Contiene la implementación de los objetos encargados de sincronizar los objetos del modelo con los DTOs y viceversa. Depende de common, model, dto, repository y persisten.

**Service-impl**: Contiene la implementación de los servicios. Depende de common, model, dto, repository, persisten, service y synchronizer. En esta capa se expondrán todas las funcionalidades del sistema.

**Web**: Contiene los archivos de SCA, y de configuración general. Depende de common, service, service-impl y dto. Se separa SCA del paquete service-impl para no generar dependencia con el framework Apache Tuscany y también para mayor organización.

Estos son los paquetes del lado servidor, del lado cliente existe un solo paquete.

**WebClient**: contiene la implementación de los objetos encargados de generar la interfaz gráfica. El cliente no tiene necesidad de utilizar los mismos paquetes de declaración de servicios y dto. Es más el cliente podría estar escrito en otros lenguajes y otras plataformas, pero para acelerar el desarrollo se utilizara los objetos dto ya desarrollados en el proyecto servidor.

El cliente esta totalmente separado de la aplicación servidor para obtener mayor desacoplamiento y a la vez esto nos permite tener diferentes clientes sin replicar código. Los clientes podrían ser aplicaciones web como la desarrollada en esta tesis, aplicaciones de escritorio, se podría publicar un post usando algún editor de texto (Microsoft word por ejemplo), readers rss, etc. Publicando las funcionalidades en diferentes protocolos permitimos que diferentes clientes puedan usar nuestra aplicación cumpliendo un requisito de la web 2.0 que es que nuestro sitio debe dar la sensación de estar en todas partes.

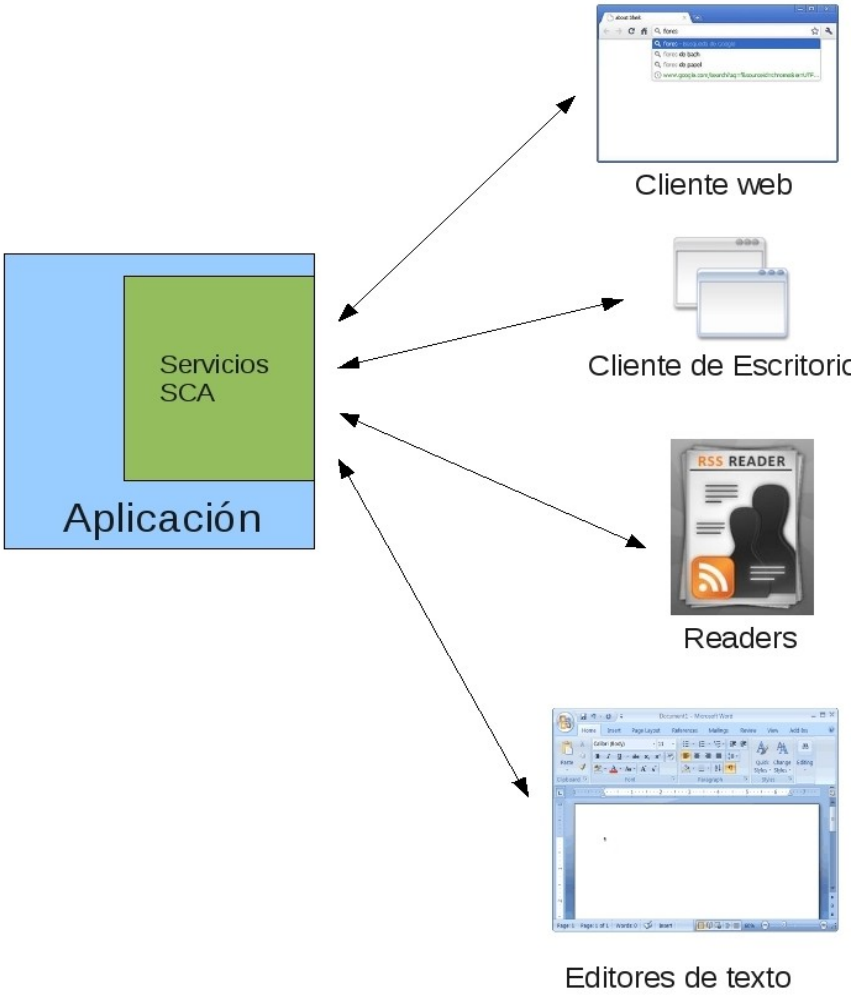


*Ilustración 16: Ejemplo de multitud de clientes*

Para comprender como interactúan los diferentes paquetes veamos un diagrama de secuencia de ejemplo :

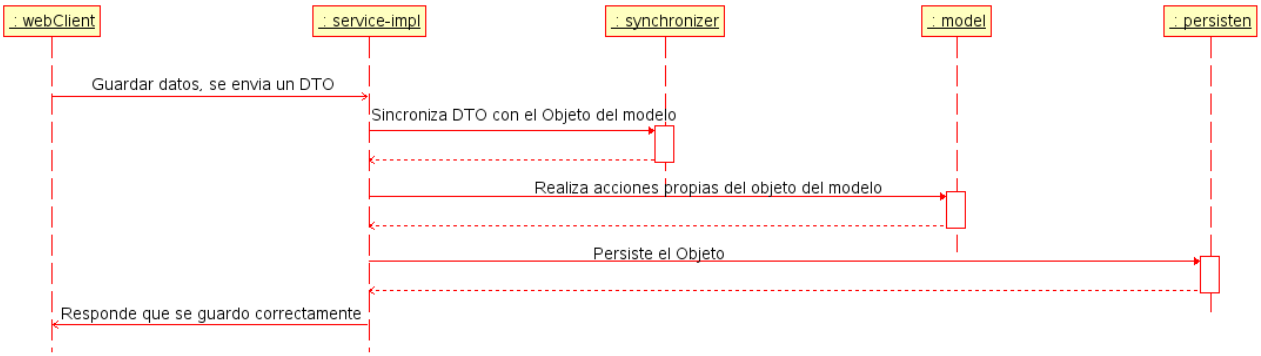


*Ilustración 17: Diagrama de secuencia*

El ejemplo muestra la siguiente secuencia:

- El cliente envía un pedido con datos de una entidad “x” para que se guarden. Lo que envía el cliente es un DTO, esto se envía por un protocolo soportado por SCA, por ejemplo SOAP.
- El servidor toma este pedido por medio de SCA se desserializa, se envía el DTO al servicio implementado que se encuentra en service-impl.
- El servicio necesita el objeto del modelo para guardar, por lo tanto debe sincronizar los datos del objeto del modelos con los del DTO, para lo que usa un objeto sincronizador.
- El objeto sincronizador, sincroniza los objetos de la siguiente manera:
    - En caso de existir un objeto con el id del DTO lo trae y aplica las modificaciones de datos que se reflejan en el DTO.
    - De lo contrario crea el objeto del modelo y lo popula o puebla con los datos del DTO.
- El servicio obtiene el objeto del modelo del proceso de sincronización, para luego permitir que el objeto del modelo ejecute métodos propios de la funcionalidad.
- Luego se llama al objeto encargado de la persistencia, este guarda el objeto.
- Por ultimo envía al cliente un mensaje, indicando el éxito de la operación.

Dado este esquema de paquetes tenemos una aplicación lo suficiente desacoplada para adaptarse a diferentes realidades y puede utilizarse como medio de comunicación en diferentes ambientes, sobre diferentes plataformas. Antes de terminar de especificar la arquitectura veamos los framework que se utilizaran para beneficiar el desarrollo.

## Framework open source y gratuitos

- **SCA** : Como se nombro anteriormente se usara como framework SCA Apache Tuscany.
- **BPEL**: WSO2 Bussiness Process Server es un producto basado en Apache ODE que como detallamos anteriormente es la mejor opción BPEL.
- **ORM**: Dado que la aplicación debe poder guardar datos en las bases de datos más utilizadas, y además debemos guardar nuestros objetos en tablas, el

mejor camino en este caso es utilizar un ORM, el cual elegimos hibernate, por su madurez y gran comunidad.

- **Inyección de Dependencias**: Spring es un framework que nos facilita el desarrollo, brindando la Inyección de dependencias, la inyección de dependencia permite que obtengamos un objeto por medio de un contexto y en este contexto configuremos las dependencias de este objeto, además Spring facilita el uso de hibernate y facilita el desarrollo de tareas relacionadas con arquitectura por ejemplo seguridad y transaccionalidad. Además se integra fácilmente con Apache Tuscany.

- **Sincronización**: Apache Dozer es un framework que facilita la sincronización de DTO con objetos del modelo.

- **Framework web**: Apache Click es un framework web joven que permite desarrollar interfaces de usuario rápidamente y de forma fácil.

- **Framework javascript**: Dado que en necesario utilizar mucho javascript para dar agilidad a el sitio; se utilizara un framework javascript llamado jquery.

Se utilizan diferentes herramientas open source que nos facilitan el desarrollo:

- **Servidor**: Las aplicaciones tanto cliente como servidor necesitan un servidor web, que corra java, existen varios servidores web open source pero sin duda los más importantes son Apache Tomcat y jetty. Por lo tanto la aplicación deberá correr y sera probada en estos dos servidores. Apache tomcat y jetty son usados por servidores de aplicación como JBoss, Apache Geronimo, JOnSA, etc. Por lo tanto la aplicación funcionaria sobre estos servidores.

- **IDE**: Como entorno de desarrollo integrado se eligió eclipse, por su agilidad y capacidad de extenderse brindando soporte a todas las actividades del desarrollo.

- **Base de datos**: Si bien la aplicación soporta cualquier tipo de base de datos para el desarrollo y las pruebas tomaremos una de referencia esta sera Postgresql y mysql dado que son las más relevantes en el mundo Open Source.

- **Herramienta de construcción**: Apache Maven provee funcionalidad de construir la aplicaciones y brinda un modelo de construcción. Con maven se puede generar proyectos y generar el ejecutable de forma fácil.

- **Herramienta de test de los web service**: Como herramienta de test de los web service se utilizara un producto llamado SoapUI, el cual se distribuye en dos

ediciones una estandar, gratuita y open source (es la que vamos a utilizar) y otra versión empresarial de pago. Los test que tenemos que realizar no son muy complejos por este motivo la  versión estándar es suficiente.

# Conclusión

El desarrollo del Blog, nos indica que se puede desarrollar aplicaciones completas con un Gobierno SOA con tecnologías abiertas y gratuitas. Existen en el mercado varios frameworks y herramientas SOA Open Source para desarrollar una infraestructura SOA. Frameworks SCA para el desarrollo de servicios como Apache Tuscany y varios productos para conformar el Gobierno SOA.

Una arquitectura basada en Open Source, si bien no tiene un respaldo de una empresa, tiene el respaldo de una comunidad. Por este motivo es importante que los productos Open Source tengan una comunidad grande y activa. Cuanto mayor y más activa sea la comunidad, mejor va ser la documentación y la calidad del software

El costo de integración de los diferentes productos Open Source es alto; se debe contar con gente capacitada y con espíritu autodidacta, la cual invertirá más tiempo que si se utiliza un producto enlatado. Pero el tiempo que se utiliza para instalar y configurar las diferentes herramientas no es tiempo perdido ya que el conocimiento de la herramienta, de su integración y de su funcionamiento pasan a ser un activo. Para capitalizar este conocimiento es importante que se documente; una buena practica es contar con una wiki en la cual se publican las lecciones aprendidas. Con una cultura que se base en resguardar el conocimiento, las herramientas open source nos dan mayor control y facilidad de uso.

Existe un abanico importante de diferentes soluciones SOA Open Source que compiten de igual a igual con productos comerciales. Además de una cantidad de importante de documentación y libros. Por estas razones es que el desarrollo SOA Open Source es viable y beneficia a quien lo usa.

La web 2.0 sobre una arquitectura SOA brinda varios beneficios, los beneficios propios del uso de SOA:

- No existirán funcionalidades replicada en varios sistemas.
- No existirán funcionalidades o procesos de integración entre sistemas.
- Mayor flexibilidad e integración.
- Bajo acoplamiento.
- Integración basada en estándares.

Y a la vez beneficios propios de su uso en la web 2.0

- Comunicación con otras web de forma fácil y estándar.
- Los sistemas están distribuidos facilitando la escalabilidad.

Como desventaja podemos afirmar que se penaliza la performance dado que se cuenta con sistemas más segmentados, la comunicación entre ellos lleva mayor tiempo, pero esta desventaja es menor dado que al ser un sistema distribuido la performance se puede aumentar aumentando el numero de servidores. SOA permite crecer de forma horizontal fácilmente, gracias a que es una arquitectura distribuida.

Otra desventaja no menor es el costo de desarrollar la web 2.0 con SOA. Las web 2.0 desarrolladas sobre una plataforma SOA son más costosas, dado que llevan más tiempo y mayor costo de hardware en la puesta en marcha del sistema. De todos modos, la inversión es justificada debido las ventajas de la plataforma. Al utilizar software Open Source y gratuito disminuye un poco el costo del proyecto.

El modelo de desarrollo SOA propuesto concuerda con los requerimientos no funcionales de una web 2.0, permitiendo desarrollar de forma fácil, un modelo arquitectónico distribuido.

SCA facilita mucho el desarrollo de aplicaciones SOA y la intercomunicación con otros productos. Y a la vez propone las mejores prácticas de programación, nos ayuda mucho a hacer las cosas bien, separando las incumbencias con lo que logramos que el modelo de objetos quede “limpio” y que no “sepa” nada de SOA. Por lo tanto el modelo es independiente de la tecnología de implementación del sistema facilitando la reutilización y posibles migraciones futuras. SCA brinda un modelo de desarrollo desacoplado y flexible. Apache tuscany es muy buena implementación, documentada y segura aunque le falta madures. Su comunidad es pequeña y el número de usuarios es poco pero tenemos que tener en cuenta que es una tecnología nueva. Apache Tuscany es un producto con mucho futuro y es esperable que su comunidad crezca.

Durante el desarrollo de esta tesis se fue gestando otro modelo arquitectónico llamado Web-oriented architecture (WOA). WOA, como SOA, tiene varias definiciones. Algunos ven a WOA como un estilo arquitectónico y un subestilo de SOA basado en aplicaciones web, más liviano que SOA. Otra definición es el uso de REST para construir servicios web usando tecnología web como HTTP y documentos XML. WOA es un estilo arquitectónico que toma las mejores características de SOA y de la Web 2.0. Este enfoque mucho más orientado a la Web es por lo que muchos lo llaman Web-Oriented Architecture (WOA) y se basa en la enorme fuerza de tracción de la World Wide Web en

sí y sus fundamentos arquitectónicos subyacentes. Basado en los conceptos básicos y los resultados que han hecho de la Web, la mayor red abierta en el planeta, así como el mayor SOA actualmente en existencia.

WOA es un modelo ideal para desarrollar SOAs livianas, lo cual encaja muy bien con una Web 2.0. Obteniendo las ventajas de SOA sin sacrificar la performance. Y a la vez es útil para desarrollar SOA empresariales.

Sin duda que si se desea desarrollar una web 2.0 es ideal poderlo hacer con WOA, pero este estilo arquitectónico comienza a gestarse por lo que no existen herramientas que lo soporten o framework que faciliten el desarrollo. A las claras WOA va a ser el camino más claro para el que quiera desarrollar una web 2.0 en un futuro.

En resumen esta tesis concluye que el desarrollo de aplicaciones web 2.0 se puede realizar con SOA Open Source y que esta arquitectura ayuda al desarrollo, brindando facilidades para satisfacer requerimientos no funcionales complejos como por ejemplo la capacidad de escalar. Además nos brinda muchos beneficios técnicos como la facilidad de comunicación con otras webs.

# Bibliografía

[PClements] **A Survey of Architecture Description Languages**

Autor: Paul Clements.

Proceedings of the International Workshop on Software Specification and Design, Alemania, 1996.

Url: http://sophia.javeriana.edu.co/~cbustaca/Arquitectura%20Software/Documentos/Descripcion/Articulos/Cle1996b.pdf

[WSArchitecture] **Web Services Architecture**

Autores: M. Champion, E. Newcomer, C. Ferris, H. Haas, D. Booth, D. Orchard, F. McCabe.

Url: http://www.w3.org/TR/2004/NOTE-wsarch-20040211/

[WikiSOA] http://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios

[AdopcionSOA] **Adopción de SOA para Dummies.**

Autores:Miko Matsumura, Bjoern Brauel y Jignesh Shah

ISBN: 978-0-470-48334-3

Publicado por Wiley Publishing.

[IBMGobiernoSOA] http://www-01.ibm.com/software/es/solutions/soa/gov/index.html

[**OpenSourceSOA**] **Open Source SOA.**

Autor: Jeff Davis

Publicado por Manning Publications corp.

ISBN: 1933988541

[WikiESB] http://es.wikipedia.org/wiki/Enterprise_service_bus

[W3CCDL] **Web Services Choreography Description Language Version 1.0. W3C Working Draft 27 April 2004.**

Url: http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/

[**IntroduccionBPM**] **Introducción a BPM para Dummies.**

Autores: Kiran Garimella, Michael Lees y Bruce Williams

ISBN: 978-0-470-37359-0

Publicado por Wiley Publishing.

[BPMandSOA] **BPM and SOA**

Autor: Mike Rosen

Url:http://www.bptrends.com/publicationfiles/04-08-COL-BPMandSOA-OrchestrationorChoreography-%200804-Rosen%20v01%20_MR_final.doc.pdf

[Complexevents] http://complexevents.com/2007/07/24/how-event-processing-improves-business-decision-making/

[WikiEAI] http://es.wikipedia.org/wiki/Enterprise_application_integration

[SoaPatterns] http://www.soapatterns.org/service_facade.php

[WikiCapas] http://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas

[IntroducingSCA] **Introducing SCA**

autor: David Chappell

Url: http://www.davidchappell.com/articles/introducing_sca.pdf

[OracleSCA] **Oracle SCA – The Power of the Composite**

Autor: Pat Shepherd

Url: http://www.oracle.com/technetwork/topics/entarch/whatsnew/oracle-sca-the-power-of-the-composi-134500.pdf

[Fabric3] **Fabric3 Reference Guide**

Url: http://dist.codehaus.org/fabric3/releases/doc/1.6/fabric3-reference-1-6.pdf

[Martinfowler] http://martinfowler.com/articles/injection.html