Universidad de las Ciencias Informáticas

Facultad 10

Título: Propuesta de arquitectura orientada a servicios para aplicaciones bajo software libre.

Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autores: Daniel González Reyes
Laynoll Díaz Martínez

Tutor: Ing. Yoandy Rodríguez Martínez

Ciudad de La Habana, Junio 2007

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ___ días del mes de Junio del año 2007.

Laynoll Díaz Martínez
Daniel González Reyes

_____

Ing. Yoandy Rodríguez Martínez

_____

Agradecimientos

A mi familia, por todo el soporte que me brinda. A Wendy, por el amor y todo el esfuerzo que le robo. A Yoandy “El Negro”, por todo. A mis amigos, porque si. A la Revolución ¿por qué no?

Daniel González Reyes

A mi familia que siempre me ha apoyado en todo y brindado la fuerza para seguir.
A mi madre en especial, que siempre ha sido faro y guía para mí.
A la memoria de mi abuela; se que con esto se hubiera sentido orgullosa.
A Yoandy, tutor y amigo, por ayudarnos tanto siempre.
A la revolución por la oportunidad de servirle.
A todos los que me apoyaron de un modo u otro, en especial a mis amigos.

Laynoll Díaz Martínez

Dedicatoria

A mis padres, que tanto hacen por mi bienestar. Nunca los defraudaré.

Daniel González Reyes

A mi familia por todo el cariño y educación que me han dado. A mi madre y mi abuela en especial, a quienes siempre tendré como mis seres más queridos.

Laynoll Díaz Martínez

Resumen

El desarrollo de nuevas tecnologías de la información se manifiesta de manera muy veloz, lo que conlleva a que las empresas constantemente actualicen y evolucionen sus aplicaciones de software. Este proceso de actualización requiere mucho tiempo, esfuerzo y capital, y no garantiza estabilidad ni eficiencia para la organización. La solución esta en implementar una arquitectura de software que defina la estructuración y funcionamiento del sistema y a la vez permita acomodarse a los cambios minimizando las situaciones problemáticas. Aunque existen muchas soluciones a la hora de promover un modelo de arquitectura que facilite esta adaptabilidad, los tiempos que corren exigen cada vez más la integración como requisito para lograr el éxito empresarial. Se necesita de estandarización, reutilización y accesibilidad entre los sistemas de software si se quiere alcanzar integración, y es aquí donde la arquitectura orientada a servicios ha logrado imponerse. La atención mundial se ha centrado en los servicios web, por lo que las organizaciones que no migran ni se proponen una solución SOA pierden indiscutiblemente posición en el mercado. Con este trabajo se persigue estudiar la trascendencia de SOA como paradigma de la arquitectura en conjunto con los servicios web, así como su situación en el mercado actual de mano de las empresas más lucrativas para proponer un modelo de arquitectura que permita desarrollar sistemas basados en la construcción, integración y consumo de servicios utilizando software libre.

PALABRAS CLAVE

Arquitectura de software, SOA, software libre, servicios web, integración, estándares

Tabla de Contenidos

Introducción

Desarrollar aplicaciones empresariales no ha sido una tarea fácil aún considerando que los avances de la tecnología han ido simplificando el manejo de varios aspectos de este problema. Los desarrolladores de este tipo de aplicaciones continúan enfrentándose a desafíos tales como datos complejos; la mayor parte del tiempo los requerimientos no son explícitos, usuarios simultáneos múltiples, los requerimientos del negocio cambian con frecuencia, plataformas heterogéneas, e interdependencias entre aplicaciones distribuidas. Además, el hecho de que no exista ningún estándar de la industria para el desarrollo de este tipo de aplicaciones ha llevado a muchas organizaciones a desarrollar sus propios modelos generando así un alto costo de desarrollo y mantenimiento de las aplicaciones.

Actualmente en las empresas el negocio cada vez exige crear aplicaciones más complejas, con menos tiempo y presupuesto. Por consiguiente se están enfocando en incrementar la flexibilidad de sus servicios y a la vez simplificar la infraestructura tecnológica utilizando servicios web.

Situación Problémica

En la Universidad de las Ciencias Informáticas existen proyectos que deben consumir servicios web de aplicaciones propias o desarrolladas por otros equipos de trabajo o proyectos, en este caso se encuentra la fábrica de portales de la Facultad 10. Este tipo de proyectos demandan una estrategia de aplicaciones empresariales que facilite su integración y además, motive la construcción de servicios más que de aplicaciones. Estos servicios se encargarían de exponer una funcionalidad bien definida a la aplicación que la requiera. De esta manera, una aplicación final simplemente orquesta la ejecución de un conjunto de estos servicios, añade su lógica particular y le presenta una interfaz al usuario final.

Problema científico

¿Qué estrategia se puede definir para motivar la integración de aplicaciones en la fábrica de portales de la Facultad 10 de la Universidad de las Ciencias Informáticas?

La presente investigación se definió como **objeto de investigación** las arquitecturas existentes de desarrollo de aplicaciones y las ventajas y desventajas del uso de las mismas, teniendo en cuenta su optimización y adaptabilidad para nuestro entorno.

Objetivo general:
Diseñar un modelo basado en una arquitectura orientada a servicios, para lograr el desarrollo e integración de aplicaciones empresariales utilizando software libre.

Objetivos específicos:

- Definir un modelo de arquitectura que permita la interoperabilidad de los servicios.
- Dirigirlo hacia una arquitectura empresarial basada en servicios.

El **campo de acción** será la arquitectura orientada a servicios para el desarrollo de aplicaciones bajo software libre.

Para el desarrollo de la investigación se proponen las siguientes **Preguntas científicas:**

- ¿Cuáles son los fundamentos de la arquitectura orientada a servicios?
- ¿Cuáles son las tecnologías software utilizadas para implementar soluciones orientadas a servicios?
- ¿Cómo diseñar un modelo de arquitectura orientada a servicios que responda a las necesidades de la fábrica de portales de la Facultad 10 de la Universidad de las Ciencias Informáticas?

Con el objetivo de guiar la investigación se trazaron las **tareas de investigación** siguientes:

- Estudiar las principales soluciones orientadas a servicios en el panorama internacional actual.
- Indagar acerca de las plataformas utilizadas para la implementación de soluciones orientadas a servicios.
- Indagar sobre una propuesta de arquitectura acorde con las condiciones de la Facultad 10 de la Universidad de las Ciencias Informáticas y que permita la implantación de la misma al menor tiempo y costo posible para la integración de servicios.

Este trabajo está estructurado en tres capítulos:

- **Capitulo 1: Fundamentación Teórica. Estado del arte.**

En este capítulo se estudiará la evolución de la arquitectura software así como los diferentes tipos de arquitectura más utilizados mundialmente, para evidenciar la necesidad actual de una arquitectura orientada a servicios.

- **Capitulo 2: Panorama Internacional. Softwares usados.**

Las soluciones SOA dominan el mercado internacional. En este capítulo se analizan las soluciones de tres de los máximos exponentes cuando de orientación a servicios se trata, Microsoft, IBM y Software AG.

- **Capitulo 3: Pautas para la integración de aplicaciones.**

En este capítulo se expone una propuesta de modelo basado en una arquitectura orientada a servicios, se explica cómo funciona este tipo de soluciones y se orientan algunas tecnologías “open source” para su implementación.

Capítulo 1: Fundamentación Teórica, estado del arte.

La necesidad del manejo de la arquitectura de un sistema de software nace con los sistemas de mediana o gran envergadura, que se proponen como solución para un problema determinado. En la medida que los sistemas de software crecen en complejidad, bien sea por número de requerimientos o por el impacto de los mismos, se hace necesario establecer medios para el manejo de esta complejidad, sin la pérdida de tiempo o la elevación de los costos y gastos de operación. Con el tiempo se han ido desarrollando metodologías y fórmulas o trucos para conseguir dichos propósitos. En la última década cambió la visión que se tiene de los sistemas de software. Esta nueva visión se llamó "Arquitectura". Desde los pequeños programas hasta los sistemas más grandes poseen una estructura y un comportamiento que los hace clasificables según su "Arquitectura". Se convirtió en el nuevo aspecto que hizo posible el estudio de sistemas ya implementados así como el desarrollo de nuevos, según la categoría del problema que resuelven y el tipo de "Arquitectura de Software" que se emplea para construirlos.

Este capítulo es el resultado de la investigación previa que se llevó a cabo para analizar el desarrollo de la arquitectura de software hasta la actualidad, sus avances, características, ventajas y desventajas de las variantes o tipos de arquitecturas existentes hasta el momento. Se exponen las características de algunas de las arquitecturas más usadas y se explica el porque la utilización de Arquitectura Orientada a Servicios para dar solución a la problemática planteada anteriormente, así como sus ventajas y desventajas para la gestión empresarial actual.

¿Qué es la Arquitectura del Software?

"La arquitectura de software de un sistema o una colección de sistemas consiste en las decisiones importantes de diseño acerca de las estructuras del software y las interacciones entre esas estructuras que conforman el sistema. Estas decisiones de diseño soportan un grupo cualidades que el sistema debería tener para lograr el éxito. Las decisiones de diseño proporcionan la base conceptual para el desarrollo, soporte y mantenimiento del sistema."(James McGovern 2003)

"La arquitectura de software de un programa o sistema de computación es la estructura o estructuras del sistema, que esta compuesto por componentes de software, las propiedades externas visibles de esos componentes, y las relaciones entre ellos."(Len Bass 1997)

"Una arquitectura es el grupo de decisiones significantes acerca de la organización de un sistema, la selección de los elementos estructurales y sus interfaces, por las cuales el sistema se compone, Además del comportamiento que se especifica en las colaboraciones entre los elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas. Conforman el estilo arquitectónico de esta organización, dichos elementos y sus interfaces, sus colaboraciones, y su composición"(Booch Grady 1998)

Existen muchas definiciones de Arquitectura del Software y no parece que ninguna de ellas haya sido totalmente aceptada. En un sentido amplio podríamos estar de acuerdo en que la Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene la responsabilidad de:

- Definir los módulos principales
- Definir las responsabilidades que tendrá cada uno de estos módulos
- Definir la interacción que existirá entre dichos módulos
- Control y flujo de datos
- Secuenciación de la información
- Protocolos de interacción y comunicación
- Ubicación en el hardware

La Arquitectura Software, también denominada Arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información. Además aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.
La definición oficial de Arquitectura del Software es la IEEE Std 1471-2000. “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”.

Usabilidad y Arquitectura del Software

"La usabilidad se trata de un sistema que permite a los usuarios cumplir con las tareas necesarias fácilmente, eficientemente, y con un mínimo de errores. Usabilidad significa que los usuarios logren sus metas con poca frustración, o ninguna."(Luke 2003)

La presentación y funcionalidad de un software son renglones de mucha importancia para que el trabajo esté completado y sea de modo correcto a un usuario final. Son dos características a medir en todo sistema para su salida, aunque no son solo estas dos características las que se deben tener en cuenta. A menudo se debe ir más lejos y no basta con tener en cuenta la presentación y la funcionalidad. Sobre todo en sistemas complejos, como pueden ser los entornos distribuidos, los transaccionales, los multicanal y aquellos en los que puede haber miles de usuarios conectados simultáneamente, hay que tener en cuenta la usabilidad desde el inicio del diseño del sistema, es decir, desde lo que se denomina momento de Arquitectura del Software.
En cualquier trabajo, mientras más tarde se detecta un problema, más cuesta arreglarlo. En estos escenarios de interacción, la causa de que no se puedan implementar es que no se tuvo en cuenta al usuario al inicio del diseño del sistema, es decir, en la Arquitectura del Software.

Los productos resultantes de la Arquitectura del Software

El objetivo principal de la Arquitectura del Software es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto. Para conseguirlo, la Arquitectura del Software construye abstracciones, materializándolas en forma de diagramas (blueprints) comentados.
No hay estándares en cuanto a la forma y lenguaje a utilizar en estos blueprints. De todas formas, existe consenso en cuanto a la necesidad de organizar dichas abstracciones en vistas, tal y como se hace al diseñar un edificio. La cantidad y tipos de vistas difieren en función de cada tendencia arquitectónica.
Quizá uno de los modelos más conocidos es el “4+1” de Philippe Kruchten, vinculado al Rational Unified Process (RUP), que define cuatro vistas diferentes:

- Vista lógica: describe el modelo de objetos.
- Vista de proceso: muestra la concurrencia y sincronía de los procesos.
- Vista física: muestra la ubicación del software en el hardware.
- Vista de desarrollo: describe la organización del entorno de desarrollo.

Existe una quinta vista que consiste en una selección de casos de uso o de escenarios que los arquitectos pueden elaborar a partir de las cuatro vistas anteriores.

Siempre que se tenga la oportunidad, sería necesario que la usabilidad se tuviera presente desde el inicio del diseño de un sistema, es decir, desde el momento de la Arquitectura del Software. Pero, para que esta participación sea realmente efectiva, arquitectos y expertos en usabilidad deberían tener un lenguaje común. Es en este sentido en el que los escenarios de usabilidad y los patrones arquitectónicos pueden ser de gran ayuda.

Nueva Concepción, adaptación a los cambios

El desarrollo de aplicaciones empresariales cada vez más exige por el cumplimiento de los requisitos de integración, seguridad, confiabilidad, escalabilidad y disponibilidad y sin el cumplimiento de estos requisitos no se puede hablar de calidad del producto desarrollado. Todos estos requisitos se deciden en el proceso de diseño de la arquitectura de la aplicación y los servicios Web tienen gran impacto en la implementación de estos requisitos.

Una buena arquitectura es una estructura del sistema estable que puede acomodarse a los cambios de requisitos y de tecnologías. Los servicios Web pueden ser vistos como la solución de almacenamiento del comportamiento que no fue debidamente alcanzado en el enfoque orientado a objetos, fundamentalmente por razones comerciales. El surgimiento de las aplicaciones web y posteriormente de los servicios web ha provocado que los métodos disponibles para la definición y análisis de la arquitectura, sean insuficientes e inapropiados para esta situación. Igualmente ocurre con el propio lenguaje “Unified Modeling Language” (UML)

Modelos o vistas

"La documentación de una arquitectura debe servir varios propósitos. Debe ser suficientemente abstracta para que sea comprendida por todos los desarrolladores, incluso los nuevos. Debe ser suficientemente detallada para servir como plano de construcción. Y debe contener suficiente información para servir de base para el análisis"(Paul Clements 2002)

Toda arquitectura software debe describir diversos aspectos del software. Generalmente, cada uno de estos aspectos se describe de una manera más comprensible si se utilizan distintos modelos o vistas. Es importante destacar que cada

uno de ellos constituye una descripción parcial de una misma arquitectura y es deseable que exista cierto solapamiento entre ellos.

Cada paradigma de desarrollo exige diferente número y tipo de vistas o modelos para describir una arquitectura. No obstante, existen al menos tres vistas absolutamente fundamentales en cualquier arquitectura:

- La visión estática: describe qué componentes tiene la arquitectura.
- La visión funcional: describe qué hace cada componente.
- La visión dinámica: describe cómo se comportan los componentes a lo largo del tiempo y como interactúan entre sí.

Las vistas o modelos de una arquitectura pueden expresarse mediante uno o varios lenguajes. El más obvio es el lenguaje natural, pero existen otros lenguajes tales como los diagramas de estado, los diagramas de flujo de datos, entre otros. Estos lenguajes son apropiados únicamente para un modelo o vista. Afortunadamente existe cierto consenso en adoptar UML como lenguaje único para todos los modelos o vistas. Sin embargo, un lenguaje general corre el peligro de no ser capaz de describir determinadas restricciones de un sistema de información (o expresarlas de manera incomprensible).

A partir de 1992 o 1993, poco después de la aparición de la arquitectura de software como especialidad profesional surgen los lenguajes de descripción de arquitecturas (Architecture Descroption Languajes). Los ADL se utilizan para satisfacer requisitos descriptivos de alto nivel de abstracción. Si se posee un ADL, un arquitecto puede analizar sobre las propiedades del sistema con precisión, pero a un nivel de abstracción convenientemente general.
Algunas de esas propiedades podrían ser, por ejemplo, protocolos de interacción, anchos de banda y latencia, localización del almacenamiento, conformidad con estándares arquitectónicos y previsiones de evolución ulterior del sistema. Estos elementos de la arquitectura tomaron gran énfasis posteriormente con el surgimiento de la tecnología de componentes.

Características de modelamiento requeridas para un Lenguaje de Descripción de Arquitecturas.

Componentes

- Interfaces
- Tipos
- Semántica
- Evolución
- Propiedades no funcionales

Conectores

- Interfaces
- Tipos
- Semántica
- Evolución
- Propiedades no funcionales

Configuraciones

- Comprensibilidad
- Composición Jerárquica
- Refinamiento y seguimiento
- Heterogeneidad
- Escalabilidad
- Evolucionalidad
- Dinamismo
- Restricciones
- Propiedades no funcionales

Componentes

Las componentes son las entidades computacionales activas de un sistema. Ellas realizan tareas mediante cómputo interno y comunicación externa con el resto del

sistema. La relación entre una componente y su entorno se define explícitamente como una colección de puntos de interacción o puertos. Los puertos se pueden generalizar como la noción de una interfaz de un módulo. En una forma simplificada, un puerto puede representar un procedimiento que puede ser llamado por otra componente, pero también puede ser un conjunto complejo de eventos que se disparan simultáneamente. Las componentes pueden ser pequeñas como un procedimiento o grandes como una aplicación entera encapsulada.

Un componente tiene sus propios datos y espacio de ejecución independientes, aunque podría compartirlos con otros componentes. Las características de un componente que un ADL debe ser capaz de modelar son las siguientes:

Interfaces

Es el conjunto de puntos de interacción entre una componente y su entorno. La interfaz especifica los servicios que provee una componente (operaciones, mensajes y variables) y también los servicios que la componente requiere de otras componentes del sistema. Es deseable que los ADL permitan definir los tipos de interfaces de las componentes de modo de maximizar la flexibilidad y la reutilización de sus definiciones.

Tipos

Los tipos de una componente son abstracciones que encapsulan funcionalidades en bloques reutilizables. Un tipo de componente puede ser instanciado múltiples veces en una sola arquitectura o ser reutilizado en diferentes arquitecturas. Los tipos de componentes pueden ser parametrizados facilitando así aún más su reutilización. El modelamiento explícito de tipos ayuda a analizar y entender una arquitectura en la que los tipos son compartidos por todas sus instancias.

Semántica

Se define la semántica como un modelo de alto nivel del comportamiento de la componente. Este modelo es necesario para realizar análisis, reforzar restricciones de arquitectura y para asegurar la consistencia entre diferentes niveles de abstracción. Los modelos semánticos pueden ir desde expresar la información semántica como una lista de propiedades de la componente hasta modelos de comportamiento dinámico.

Restricciones

Las restricciones son propiedades o afirmaciones sobre un sistema o alguna de sus partes. Una componente puede ser restringida mediante atributos no funcionales, por ejemplo: Tiempo de ejecución o “deadlines”.

Evolución

Los ADL pueden permitir la evolución de componentes definiendo tipos de componentes y permitiendo el refinamiento mediante subtipado (subtyping).

Propiedades no funcionales

Estas propiedades no tienen relación con la funcionalidad sino con características de calidad de la componente tales como seguridad, rendimiento y portabilidad.

La mayoría de los ADL tiene a las componentes como sus principales elementos de especificación; todos modelan interfaces y distinguen entre tipos e instancias de componentes. Por otro lado, casi ningún ADL soporta evolución ni especificación de propiedades no funcionales.

Conectores

Los conectores definen la interacción entre los componentes. Cada conector provee una forma para que una colección de puertos esté en contacto y define lógicamente el protocolo a través del cual un conjunto de componentes puede interactuar. Los puertos definen los puntos de interacción de los conectores. Al igual que las componentes, un conector tiene una interfaz, la cual consiste en un conjunto de roles. Cada rol define el comportamiento esperado de uno de los participantes en la interacción. El comportamiento total de un conector está definido por un protocolo. Por lo tanto las características principales que debe tener un conector son las siguientes:

Interfaz

Es el conjunto de puntos de interacción entre un conector y una componente u otro conector. Las interfaces de conectores permiten una conectividad apropiada entre componentes y su interacción en una arquitectura. En general, cuando un ADL soporta conectores como entidades de primera clase, entonces soporta explícitamente también la especificación de las interfaces de estos conectores.

Tipos

Los tipos de conectores son abstracciones que encapsulan la comunicación, coordinación y decisiones de mediación de componentes.

Semántica

La semántica de conectores se define como un modelo de alto nivel del comportamiento de un conector. La semántica expresa funcionalidad a nivel de la aplicación y la especificación de protocolos de interacción.

Restricciones

Las restricciones de los conectores aseguran la adherencia a protocolos de interacción. Además establecen dependencias entre conectores y refuerzan el uso de límites.

Evolución

La evolución de un conector se refiere a la modificación de las propiedades de un conector, o sea, de la modificación de su interfaz, semántica, o restricciones entre las dos.

Propiedades no funcionales

Las propiedades no funcionales de los conectores no se pueden derivar completamente de la especificación de su semántica. Permiten simulaciones de comportamiento en tiempo de ejecución y reforzamiento de restricciones.

Distintos ADL modelan conectores de variadas formas y con variados nombres. Muchos modelan conectores explícitamente y los llaman conectores, y otros como servicios de transporte.

Configuraciones

Una configuración o topología es una colección de instancias de componentes que interactúan mediante instancias de conectores. En otras palabras, una topología es un grafo de componentes y conectores conectados que describen la estructura de la arquitectura. Las características de nivel de configuración se agrupan en tres categorías generales:

- *Calidad de la descripción de la configuración:*

Entendimiento, composición, refinamiento y seguimiento, heterogeneidad.

- *Calidad de Descripción del sistema:*

Heterogeneidad, escalabilidad, evolución y dinamismo.

- *Propiedades de la descripción del sistema:*

Dinamismo, restricciones y propiedades no funcionales.

Parámetros que forman parte de las configuraciones

Especificaciones entendibles

Uno de los roles de la arquitectura de software es el de servir como conducto de comunicación y así facilitar el entendimiento del sistema y la abstracción de alto nivel. Por lo tanto los ADL deben ser capaces de modelar información estructural con una sintaxis comprensible.

Composición jerárquica

Es un mecanismo que permite que una arquitectura sea definida con distintos niveles de detalle. Estructuras y comportamiento complejos pueden ser representados explícitamente o abstractamente como una composición de componentes y conectores simples.

Refinamiento y seguimiento

Los ADL deben proveer refinamiento de arquitecturas en sistemas ejecutables y seguimiento de los cambios a través de los distintos niveles.

Heterogeneidad

Las arquitecturas de software deben facilitar el desarrollo de sistemas de gran escala mediante la existencia de componentes y conectores de distinta granularidad, posiblemente especificadas en diferentes lenguajes de modelamiento e implementadas en diferentes lenguajes de programación.

Escalabilidad

Los ADL deben soportar la especificación y el desarrollo de sistemas de gran escala que pueden crecer en el futuro.

Evolucionabilidad

Una arquitectura evoluciona para reflejar y permitir la evolución de familias de sistemas de software.

Dinamismo

Se refiere a modificar la arquitectura y reflejar esas modificaciones en el sistema mientras éste se ejecuta. Es importante el soporte de dinamismo en sistema como control de tráfico, en donde la disponibilidad y seguridad son críticas.

Restricciones

Restricciones que representan dependencias en una configuración complementan aquellas específicas a las componentes y conectores.

Propiedades no funcionales

Algunas de las propiedades no funcionales están a nivel del sistema más que a nivel de componentes y conectores individuales. Propiedades no funcionales a nivel de sistema son necesarias para seleccionar apropiadamente componentes y conectores, realizar análisis y reforzar restricciones, entre otras.

Otras características deseables

Otra de las características más deseables de un ADL es que existan herramientas asociadas de apoyo tanto para hacer análisis como para hacer simulación de la ejecución de las arquitecturas especificadas. Si bien esta característica no es intrínseca al ADL, es algo a tener en cuenta a la hora de hacer una elección ya que esto afecta enormemente su usabilidad y la robustez de las especificaciones puede asegurarse en una etapa más temprana.

UML no es un ADL en el sentido usual de la expresión. Los manuales contemporáneos de UML carecen del concepto de estilo y sus definiciones de “arquitectura” no guardan relación con lo que ella significa en el campo de los ADL.

Método de definición de la arquitectura

El método propuesto se basa en la utilización de UML ya que, a pesar de las desventajas mencionadas anteriormente de este lenguaje y aunque no es un ADL, es la opción más adecuada a los efectos industriales en este momento por las siguientes razones:

- Constituye el estándar internacional para la modelación de sistemas.
- Se poseen herramientas CASE
- .Permite la modelación total del sistema y no solo de la arquitectura.

La definición de la Arquitectura es una actividad de diseño que se realiza de forma iterativa e incremental. Esto se hace en los ciclos o iteraciones de la Etapa de Elaboración. Usualmente en uno o dos ciclos de la Elaboración se realizan los casos de uso que determinan la arquitectura de la aplicación y por tanto en el diseño de esos dos ciclos se realiza la definición de la arquitectura de la aplicación. Esta actividad es realizada por el Arquitecto de la Aplicación de forma independiente de los Analistas y otros diseñadores.

Los pasos son:

- Definición de los casos de uso más significativos para la arquitectura
- .Definición del patrón de arquitectura
- .Definición del modelo de mini arquitectura (framework) a emplear
- .Identificación de los subsistemas
- .Definición de servicios web
- .Definición del Diagrama de subsistemas funcionales
- .Definición de las clases y paquetes más significativos
- .Definición de la dinámica de la aplicación.

Diseño arquitectónico: El proceso de diseño inicial para identificar los subsistemas y establecer un marco de trabajo para el control y la comunicación de los subsistemas.

La documentación de la arquitectura esta constituida por:

- Modelo Estructural estático. *Muestra subsistemas o componentes.*
- Modelo del proceso dinámico. *Muestra la organización del sistema en ejecución*
- .Modelo de la interfaz. *Define los servicios de cada subsistema*
- .Modelo de relación. *Muestra las relaciones entre subsistemas.*

Acorde con la documentación a obtener las actividades del diseño de la arquitectura son:

- Estructuración del sistema
- .Modelado del control
- .Descomposición modular

.

*Estructuración***:** El sistema se estructura en varios subsistemas principales, donde un subsistema es una unidad de SW independiente. Se identifica la comunicación
Modelado del control: Se establece un modelo general de las relaciones de control entre las partes del sistema
Descomposición modular: Cada subsistema identificado se descompone en módulos. El arquitecto debe decidirse sobre los tipos de módulos y sus interconexiones.

Arquitecturas más comunes

Generalmente, no es necesario inventar una nueva arquitectura software para cada sistema de información. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto. Así, las arquitecturas más universales son:

- Monolítica. Donde el software se estructura en grupos funcionales muy acoplados.
- Cliente-servidor. Donde el software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones.
- Arquitectura de tres niveles. Especialización de la arquitectura cliente-servidor donde la carga se divide en tres partes con un reparto claro de funciones: una capa para la presentación, otra para el cálculo y otra para el almacenamiento. Una capa solamente tiene relación con la siguiente.

Otras arquitecturas menos conocidas son:

- En pipeline.
- Entre pares.
- En pizarra.
- Orientada a servicios.
- Máquinas virtuales.

Existen diversos puntos de vista sobre la manera en que debería efectuarse el procesamiento de datos, aunque la mayoría que opina, coincide en se encuentra en un

proceso de evolución que se prolongará todavía por algunos años y que cambiará la forma en que se obtiene y utiliza la información almacenada electrónicamente.

El principal motivo detrás de esta evolución es la necesidad que tienen las organizaciones (empresas o instituciones públicas o privadas), de realizar sus operaciones más ágil y eficientemente, debido a la creciente presión competitiva a la que están sometidas, lo cual se traduce en la necesidad de que su personal sea más productivo, al mismo tiempo que se generan productos y servicios más rápidamente y con mejor calidad.

En este contexto, es necesario establecer una infraestructura de procesamiento de información, que cuente con los elementos requeridos para proveer información adecuada, exacta y oportuna en la toma de decisiones y para proporcionar un mejor servicio a los clientes.

Arquitectura Monolítica

Desde sus inicios el modelo de administración de datos a través de computadoras se basaba en el uso de terminales remotas, que se conectaban de manera directa a una computadora central. Dicha computadora central se encargaba de prestar servicios caracterizados por que cada servicio se prestaba solo a un grupo exclusivo de usuarios.

Los primeros usos de los computadores como “máquinas de cálculo” van dejando paso a otras aplicaciones. Menos cálculo y más “procesamiento de información” es requerido. Los “mainframes” evolucionan para almacenar información. Aparece la noción de “fichero” o “archivo”, como base fundamental para almacenar datos en “memoria secundaria”. Nacen usuarios y aplicaciones que quieren “recuperar” y “modificar” la información almacenada. Surge la idea de “trabajo transaccional”. Los usuarios y las aplicaciones actualizan y consultan los datos. Las aplicaciones intentan automatizar procesos diarios de las organizaciones y manejan la información de éstas.

La máxima de la integración y unificación de los datos sugiere, en un primer lugar:
Unificar e integrar todos los datos de la organización (o de un área de la organización) en un solo ordenador.

Si sólo tenemos un usuario y una aplicación, la solución es sencilla:

- Centralizamos todos los datos en un mismo ordenador.
- Construimos una aplicación que trabaje sobre los datos en el mismo ordenador.
- La organización de los datos se especializará para la aplicación.

Multiusuario, multiaplicación, trabajo transaccional:
La interfaz de la máquina central (mainframe) no se puede usar para todos los usuarios. Aparecen “terminales” y el “teleproceso”. Aparecen los problemas clásicos de gestión de datos:

- Integridad.
- Acceso concurrente.
- Seguridad.
- Independencia.

Los primeros Sistemas Gestores de Bases de Datos (SGBD) y modelos de datos aparecen con el objetivo de resolver los problemas anteriores. Surge o aparece entonces la Arquitectura Monolítica de Bases de Datos.

Usos:

- Sistemas heredados con esta arquitectura (los terminales suelen ser ahora PCs emulando un terminal).
- Terminales de vídeo y PPV.
- Terminales móviles (WAP,..).

Problemas de la arquitectura monolítica

- Cuellos de botella:
- La red y el sistema de control de terminales.
- El propio sistema central debe ser un gran “mainframe”.
- Si existe gran diversidad de aplicaciones y gran número de usuarios el servidor se satura.
- Si la visualización es compleja (gráfica) y necesita refresco inmediato la red se satura.

- Fiabilidad:
- Si el sistema central falla, falla todo, incluso las aplicaciones que no usan datos o que no los usan frecuentemente.
- El coste del sistema central es alto y no es posible sustituirlo frecuentemente por lo que los problemas se agravan.

Existen muchísimas situaciones así hoy en día:

- Ordenadores personales con datos en local.
- Sistemas de bases de datos monopuesto y monousuario: muchas aplicaciones pequeñas en Access son de este estilo.

En este modelo, el sistema operativo y todos los servicios fundamentales, tales como file system (sistema de archivos), residen en un monitor monolítico que se accede a través de un mecanismo de llamada al núcleo. Las llamadas al núcleo hacen la transición del modo aplicación (o usuario) al modo supervisor. Se proporciona protección de modo que solo se puede acceder a recursos del núcleo en modo supervisor; las aplicaciones se ponen en ejecución generalmente como procesos que tienen espacios de direccionamiento separados con protección entre ellos. Este modelo es adoptado para corregir algunos de los problemas de la configuración "ejecutivo en tiempo real".

Con un monitor monolítico, los servicios fundamentales que requieran acceso a los recursos del núcleo deben residir en este. De esta forma, la complejidad del núcleo aumenta, aumentando la probabilidad de encontrar errores. Asimismo, el acceso a entrada-salida, al vector de interrupciones, y a la memoria física se puede restringir al núcleo por razones de la seguridad, lo que significa que la mayoría de los “drivers” de dispositivos deben residir en el núcleo. Cruzar la barrera kernel/aplicación es con frecuencia costoso, así pues, en algunos casos, los “drivers” que de otro modo se podrían poner en ejecución modo usuario, tal como “drivers” de gráficos, se incorpora en el núcleo por razones de rendimiento. Mientras que las aplicaciones no pueden corromper el núcleo, cualquiera de estos “drivers” de dispositivos puede, aumentando así, la probabilidad de que ocurran errores fatales.

Arquitectura Cliente/Servidor

El modelo Cliente/Servidor reúne las características necesarias para proveer esta infraestructura, independientemente del tamaño y complejidad de las operaciones de las organizaciones públicas o privadas y, consecuentemente, desempeña un papel importante en este proceso de evolución. sta arquitectura consiste básicamente en que un programa, el Cliente informático realiza peticiones a otro programa, el servidor, que les da respuesta.

Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema multiusuario distribuido a través de una red de computadoras. En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa.

Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema. La arquitectura cliente/servidor sustituye a la arquitectura monolítica en la que no hay distribución, tanto a nivel físico como a nivel lógico.

Ventajas de la arquitectura Cliente/Servidor

- *Centralización del control:* los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.
- *Escalabilidad:* se puede aumentar la capacidad de clientes y servidores por separado.
- *Reducción considerable del tráfico de red:* Idealmente, el cliente se comunica con el servidor utilizando un protocolo de alto nivel de abstracción como por ejemplo SQL

Arquitectura de tres niveles

La programación por capas es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico es separar la capa de datos de la capa de presentación al usuario. La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación seria: Modelo de interconexión de sistemas abiertos

Además permite distribuir el trabajo de creación de una aplicación por niveles, de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, simplemente es necesario conocer la API que existe entre niveles. En el diseño de sistemas informáticos actual se suele usar las arquitecturas multinivel o Programación por Capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

Capas o niveles

- *Capa de presentación*: es la que ve el usuario, presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio.
- *Capa de negocio*: es donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.
- *Capa de datos*: es donde residen los datos. Está formada por uno o más gestor de bases de datos que realiza todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Todas estas capas pueden residir en un único ordenador (no sería lo normal), si bien lo más usual es que haya una multitud de ordenadores donde reside la capa de presentación (son los clientes de la arquitectura cliente/servidor). Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o más ordenadores. Así, si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores los cuales recibirán las peticiones del ordenador en que resida la capa de negocio.

Si por el contrario fuese la complejidad en la capa de negocio lo que obligase a la separación, esta capa de negocio podría residir en uno o más ordenadores que realizarían solicitudes a una única base de datos. En sistemas muy complejos se llega a tener una serie de ordenadores sobre los cuales corre la capa de datos, y otra serie de ordenadores sobre los cuales corre la base de datos.

En una arquitectura de tres niveles, los términos "capas" y "niveles" no significan lo mismo ni son similares. El término "capa" hace referencia a la forma como una solución es segmentada desde el punto de vista lógico:
- Presentación/ Lógica de Negocio/ Datos.

En cambio, el término "nivel", corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física. Por ejemplo:

- Una solución de tres capas (presentación, lógica, datos) que residen en un solo ordenador (Presentación+lógica+datos). Se dice, que la arquitectura de la solución es de tres capas y un nivel.
- Una solución de tres capas (presentación, lógica, datos) que residen en dos ordenadores (presentación+lógica, lógica+datos). Se dice que la arquitectura de la solución es de tres capas y dos niveles.
- Una solución de tres capas (presentación, lógica, datos) que residen en tres ordenadores (presentación, lógica, datos). La arquitectura que la define es: solución de tres capas y tres niveles.

El presente trabajo pretende centrar el estudio en la Arquitectura Orientada a Servicios debido a las ventajas que esta proporciona a la gestión empresarial. Esta arquitectura es de las menos conocidas debido a su reciente aparición dentro de la veloz evolución

de las tecnologías, no obstante, parece ser una solución eficaz a los requerimientos empresariales actuales.

En las últimas décadas, los departamentos de IT (“Information Technologies”, Tecnologías de la Información) de las empresas han venido construyendo una infraestructura que actualmente soporta en gran medida la operación de sus empresas. El resultado de este proceso, ha sido la creación y mantenimiento de un número considerable de aplicaciones al interior de las empresas, cada una responsable de sus propias tareas.

El negocio cada vez exige crear aplicaciones más complejas, con menos tiempo y presupuesto que antes. Crear estas aplicaciones, requiere en muchos casos de funcionalidades ya implementadas como parte de otros sistema. En este punto los arquitectos de soluciones tienen dos opciones:

- Tratar de reutilizar la funcionalidad ya implementada en otros sistemas. Una labor difícil de realizar, debido a que estos no fueron diseñados para integrarse sobre plataformas y tecnologías incompatibles entre ellas. Incluso en el caso de encontrarse la manera técnica de realizar la conexión, se debe asumir el riesgo de alterar un sistema en producción que esta funcionando sin problemas.
- Reimplementar la funcionalidad requerida ("reinventar la rueda"). Aunque implica más tiempo de desarrollo, es la más fácil y segura.

Por su facilidad y aunque no es la más acertada a largo plazo, la segunda opción es la más escogida. Esto trae como resultado:

- Dificultad de migración de los sistemas internos. Al haber múltiples conexiones desde sistemas que dependen de estos para su funcionamiento.
- Al no haber una estrategia de integración de aplicaciones, se generan múltiples puntos de falla, que pueden detener la operación de todos los sistemas muy fácilmente.
- Un modelo así, por lo general no escala muy bien.
- El inconveniente final es una pobre respuesta al cambio. Las aplicaciones siguen siendo concebidas desde un principio como islas independientes.

Es entonces donde comienza la acción de SOA para resolver este problema y crear un mejor funcionamiento de este tipo de cambio en el sistema de aplicaciones

empresariales. Se debe motivar la construcción de servicios en lugar de aplicaciones, los cuales se encargarían de exponer una funcionalidad bien definida a la aplicación que la requiera. Así, una aplicación final simplemente orquesta la ejecución de un conjunto de estos servicios, añade su lógica particular y a través de una interfaz es presentada usuario final.

Arquitectura orientada a servicios

La Arquitectura Orientada a Servicios (SOA: “Service Oriented Architecture”), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

Una arquitectura orientada a servicios es un método de diseñar y construir soluciones de software poco acopladas que expongan sus funciones del negocio como servicios de software accesibles a través de programación para ser usadas por otras aplicaciones a través de la publicación de interfaces. Los servicios web representan una implementación de una arquitectura orientada a servicios pero no todas las aplicaciones SOA pueden ser consideradas servicios web.

En un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. La mayoría de las definiciones de SOA identifican la utilización de servicios web (empleando SOAP y WSDL) en su implementación, no obstante se puede implementar una SOA utilizando cualquier tecnología basada en servicios.

Al contrario de las arquitecturas orientadas a objetos, las SOA están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación. La definición de la interfaz encapsula (oculta) las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo (como Java o .NET). Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar; así, un servicio “C Sharp” podría ser usado por una aplicación Java.

Los lenguajes de alto nivel como BPEL (Lenguaje de Ejecución de Procesos de Negocio) o “WS-coordination” llevan el concepto de servicio un paso adelante al proporcionar métodos de definición y soporte para flujos de trabajo y procesos de negocio.

La mayoría de las personas tiene las siguientes suposiciones básicas sobre una SOA:

- Que no hay necesidad de “romper y reemplazar” los sistemas existentes.
- Que la SOA y las normas de los servicios de Web ayudan a paliar el dolor de la integración.
- Que la SOA permite la reutilización de aplicaciones existentes (hechas y empaquetadas dentro de la empresa).

La reutilización es uno de los factores motivadores y de éxito de una SOA. Sin embargo, ¿qué significa reutilizar en el contexto de la SOA? ¿Denota la reutilización de servicios de Web de reciente creación únicamente? ¿Acaso también comprende la reutilización de aplicaciones existentes? ¿Comprende la reutilización tanto de las mejores prácticas de programación como de tecnologías y de directrices para el desarrollo de software existentes? Y además, ¿Cómo se logra la reutilización?
Muchas organizaciones que se han aventurado por los caminos de la SOA han llegado a darse cuenta de que lograr una verdadera reutilización no es tan sencillo como crear servicios web y ponerlos a disposición del consumo. Muchos factores diferentes complican el simple acto de hacer uso de los servicios que se han puesto a disposición de los usuarios dentro de una empresa.
Entender la naturaleza de la reutilización dentro del contexto de una SOA y la manera de generar la adquisición son pasos claves en el entendimiento de la promesa que encierra la SOA.

¿Qué es la Reutilización?

La reutilización puede adoptar muchas formas distintas dentro del contexto de SOA. Una definición sencilla del término es la capacidad que diversas personas o consumidores de servicios tienen para utilizar una y otra vez el mismo servicio, componente, procedimiento, directriz o proceso para realizar una tarea determinada. A partir de dicha definición está claro que la reutilización va más allá del mero uso de los servicios web o de aplicaciones existentes expuestas como servicios web. Esta definición también involucra otras áreas y disciplinas centrales que integran una cultura de colaboración, lo que proporciona los medios para crear una verdadera capacidad de reutilización de los servicios.

Obstáculos para la Reutilización

Con frecuencia, la reutilización de los servicios es más complicada en la práctica que en la teoría. Existen muchas razones para ello. Entre los problemas más frecuentes se encuentran los siguientes:

Falta de apoyo al nivel de la organización:

Al momento de instrumentar una SOA, usted necesita el apoyo tanto de los desarrolladores como de los administradores. El desafío mayor no siempre tiene que ver con problemas tecnológicos. El desafío mayor generalmente lo constituye la cultura de negocios.

La implantación de una SOA trae implícita la idea del cambio, de muchísimos cambios en ocasiones, y la gente reacciona al cambio en diferentes maneras.

Hasta cierto punto, todos somos resistentes al cambio, especialmente cuando no se nos ha incluido en una decisión que significa un impacto para nosotros o cuando tal vez no estamos de acuerdo con un enfoque en particular.

De igual forma, es probable que una iniciativa SOA fracase si no cuenta con el apoyo y la participación de los arquitectos de TI, los administradores y los equipos de desarrollo. Es mucho más sencillo alcanzar las metas de reutilización de una SOA cuando los equipos de desarrollo están comprometidos en la estrategia adecuada y cuando siguen las directrices y las mejores prácticas establecidas.

Interoperabilidad: Los problemas de interoperabilidad dificultan la interacción entre los consumidores y los productores de servicios. Con frecuencia, un consumidor puede interactuar con un servicio Web que utilice el estilo de asignación RPC. Sin embargo, el consumidor no puede interactuar con un servicio web similar que se exponga utilizando un estilo de asignación Documento/Literal. En ocasiones, los consumidores son incapaces de manejar tipos de datos complejos creados por la exposición de una aplicación heredada.

Ausencia de estándares: La ausencia de estándares puede llevar a enfoques en conflicto respecto a la manera de crear, exponer y consumir servicios. Algunos desarrolladores suponen que se tienen que utilizar servicios web al momento de crear una SOA. Sin embargo, los servicios web no son obligatorios. Otros desarrolladores prefieren utilizar XML o XML RPC, o incluso el estilo REST de los servicios web. Esto puede llevar a muchos problemas de índole práctico al momento de instrumentar una SOA.

Descubrimiento de servicios: Muchas organizaciones no cuentan con medios efectivos para determinar qué servicios existen y la manera de acceder a ellos. Para evitar esto, nosotros promovemos un enfoque de administración estructurado al igual que una ubicación centralizada para almacenar todos los metadatos y artefactos relacionados con los servicios dentro de la SOA.

Comunicación deficiente: La adquisición de una SOA se basa en la capacidad de integrar aplicaciones basadas en silos. Para que las aplicaciones se comuniquen en forma más eficiente, la organización de TI también tiene que adoptar la manera en la que se comunica.

La SOA requiere niveles de comunicaciones mejorados, que se pueden facilitar mediante un equipo de funciones entrelazadas constituido por personas que representen diferentes áreas dentro de la organización. Este grupo nuclear (o "Competency Center") puede ser un elemento importantísimo para establecer y adoptar las mejores prácticas y estándares corporativos. Sin embargo, en primer lugar y ante todo, la función de un grupo de este tipo es compartir sus ideas, experiencias y conocimientos.

Definiciones

Término	**Definición / Comentario**
Servicio	Una función sin estado, auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo como serían editar y procesar una transacción. Los servicios no dependen del estado de otras funciones o procesos. La tecnología concreta utilizada para prestar el servicio no es parte de esta definición.
Orquestación	Secuenciar los servicios y proveer la lógica adicional para procesar datos. No incluye la presentación de los datos. Coordinación.
Sin Estado	No mantiene ni depende de condición pre-existente alguna. En una SOA los servicios no son dependientes de la condición de ningún otro servicio. Reciben en la llamada toda la información que

	necesitan para dar una respuesta. Debido a que los servicios son "sin estado", pueden ser secuenciados (orquestados) en numerosas secuencias (algunas veces llamadas tuberías o pipelines) para realizar la lógica del negocio.
Proveedor	La función que brinda un servicio en respuesta a una llamada o petición desde un consumidor.
Consumidor	La función que consume el resultado del servicio provisto por un proveedor.

Diseño y desarrollo

La metodología de modelado y diseño para aplicaciones SOA se conoce como análisis y diseño orientado a servicios. La arquitectura orientada a servicios es tanto un marco de trabajo para el desarrollo de software como un marco de trabajo de implantación. Para que un proyecto SOA tenga éxito los desarrolladores de software deben orientarse ellos mismos a esta mentalidad de crear servicios comunes que son orquestados por clientes o “middleware” para implementar los procesos de negocio. El desarrollo de sistemas usando SOA requiere un compromiso con este modelo en términos de planificación, herramientas e infraestructura.

Cuando la mayoría de la gente habla de una arquitectura orientada a servicios están hablando de un juego de servicios residentes en Internet o en una intranet, usando servicios web. Hay un juego de estándares de los que se habla ligados a los servicios web. Incluyen los siguientes:

- XML
- HTTP
- SOAP
- WSDL
- UDDI

Hay que considerar, sin embargo, que un sistema SOA no necesariamente necesita utilizar estos estándares para ser "orientado a servicios" pero es altamente recomendable su uso.

Conceptos básicos

De acuerdo a analistas de la industria, los conceptos básicos de la Arquitectura Orientada a Servicios (SOA) se establecieron desde hace 20 años. Por tanto, ¿qué ofrece de nuevo?, ¿por qué esta tecnología tiene tanto éxito mientras otras fallan? ¿Cuáles son los elementos del SOA más importantes para su éxito?

Como primer punto se encuentra la flexibilidad. SOA es la primera arquitectura de tecnologías de la información (TIC) que asume lo que los negocios han sabido desde hace mucho tiempo. Se trata esencialmente de un grupo de servicios sueltos, donde cada uno es relativamente económico para construirlo o reemplazarlo si es necesario. Al ser independientes, el poder unirlos permite a SOA adaptar cambios, cuestión imposible para arquitecturas tradicionales.

En la arquitectura orientada a servicios, se puede reemplazar un servicio sin tener que preocuparse por la tecnología fundamental; la interfase es lo que importa, y está definida en un estándar universal en servicios web y XML. Esto es flexibilidad a través de la interoperabilidad. También es la habilidad de asegurar los activos existentes, aplicaciones y bases de datos legales y hacerlos parte de las soluciones empresariales extendiéndolos al SOA en vez de reemplazarlos. El resultado en la red es la habilidad de evolucionar rápida y eficientemente, en otras palabras, adaptarse "orgánicamente" de acuerdo a la demanda del negocio. Esto es realmente nuevo.

En segundo lugar está la relevancia para el negocio. SOA es TIC expresada a un nivel que tiene un significado importante para la colaboración del negocio y profesionales del área. Sus servicios actuales pueden coordinar unidades de trabajo muy cercanas a las actividades del negocio; piense, por ejemplo, en un servicio llamado "Actualización de órdenes de trabajo". Éstos son inmediatamente relevantes para los analistas de la empresa que participan en la creación y definición de nuevos procesos permitiendo el "Servicio Dirigido Empresarial".

Desde que los servicios web sustituyen la mayoría de las tecnologías fundamentales, muy poca tecnología de habla es requerida. Los negocios y las TIC se enfocan en la lógica del negocio y la comunicación; finalmente comparten el lenguaje de servicios. Esto también es relativamente nuevo y tendrá implicaciones en la entrega de servicios TIC.

Barreras a vencer para obtener el éxito con SOA

SOA es un nuevo horizonte para las TIC. Como cualquier gran cambio, las principales barreras son organizacionales, no técnicas. A continuación se ejemplifican algunas:

- *Administración:* Servicios compartidos es lo principal para utilizar SOA. La habilidad para ensamblar rápidamente aplicaciones o procesos está basada en la disponibilidad de algunos servicios que pueden ser compartidos. Hacer esto, por definición, requiere administración.
- *Desarrollo Cultural:* Al utilizar SOA se requiere un cambio significativo en el estilo de programar. Muchos desarrolladores utilizan equipos diferentes para resolver problemas de manera independiente para cada aplicación. En SOA necesitarán escribir aplicaciones para ser re-utilizadas en mente, usando códigos existentes, a los cuales se podrá tener acceso constantemente.

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. Es una metodología bastante simple, cumple con los estándares de programación más estrictos, logra aplicaciones robustas y siempre logrando que los sistemas ya sean para la web o locales sean tan sencillos de usar que no necesiten manual de operación y sean a su vez rápidos y eficientes.

En la actualidad no se tiene límite en cuanto a plataforma de uso, puede usarse y desarrollarse sobre Linux, Windows, Mac OS, entre otros. Se puede acceder de una manera segura y desde cualquier parte del mundo si así lo requieren sus necesidades. El surgimiento de la arquitectura orientada a servicio está representando un cambio de paradigma de software. En la actualidad el factor que determina la eficiencia de una empresa es su agilidad y capacidad para adaptarse a los cambios del mercado. La complejidad de las tecnologías de la información, en muchos casos, no permite que estos cambios se realicen con la agilidad requerida. La importancia de SOA como herramienta para lograr una eficiente integración radica precisamente en lo mencionado anteriormente.

La arquitectura orientada a servicios permite automatizar, estructurar y monitorear los procesos de negocios más importantes; reutilizar e integrar las aplicaciones existentes, así como crear la una nueva generación de aplicaciones. Todo ello, con la finalidad de reducir el tiempo de respuesta al mercado, incrementar la eficiencia, flexibilidad y

agilidad de los procesos, así como disminuir el costo global de integración de sistemas.

SOA es un tipo de arquitectura cuyo objetivo principal es lograr un débil acoplamiento entre los componentes de software que interactúan entre sí. La idea del débil acoplamiento esta enfocada a que cada componente del sistema no este atado el uno al otro, de forma tal que se puedan intercambiar fácilmente, en consecuencia lograr una interacción más flexible entre componentes. Por otro lado, atomicidad hace referencia a que se ejecute una tarea en una unidad de trabajo, en consecuencia una tarea se ejecuta de principio a fin sin que sea intervenida por otra.

Los elementos básicos que conforman SOA son:

- Proveedores de Servicios.
- Consumidores de Servicios.
- Bus Empresarial de Servicios.

La integración de aplicaciones es el mayor reto de la actualidad para muchos negocios. Un “Enterprise Service Bus” (ESB) es probablemente la vía más rápida y menos costosa para hacer frente a este desafío.

En realidad todo componente dentro de una organización puede ser tanto proveedor como consumidor de servicios. Todos los servicios interactúan entre si a través de un Bus Empresarial de Servicios. El ESB es probablemente el componente más importante aún no mencionado. El ESB se basa en la mejor práctica en patrones de diseño para integración de aplicaciones.

Cuando se habilitan o se agregan interacciones de servicios web que usan la infraestructura existente en Internet o alguna intranet entre sistemas en una arquitectura, se pueden crear muchas integraciones punto a punto (point-to-point) individuales, las cuales son difíciles de mantener. Como las organizaciones se mueven en torno a arquitecturas orientadas a servicios, las cuales hostean varios servicios desplegados, estas requieren una infraestructura de servicios que provea una robusta comunicación, enrutamiento inteligente y una sofisticada traducción y transformación de servicios.

El despliegue de SOA requiere la conversión de los sistemas existentes en servicios. Las tareas envueltas en alcanzar esto pueden estar repetidas por cada sistema y una serie común de componentes puede necesitarse para brindar funcionalidades adicionales (seguridad y control). Un ESB es una colección de servidores y componentes que proveen estas funciones y una serie de herramientas para ayudar en la conversión de los sistemas existentes a servicios. El ESB debe alcanzar esto sin entrar en conflictos con los principios de SOA.

Resuelve los problemas de integración en la colocación de transporte por soporte basados en estándares del mismo como HTTP. Además soporta una variedad de transportes usados comúnmente como “WS-ReliableMessaging” y “MQSeries”, y sistemas de legado como carpetas compartidas, así los sistemas existentes pueden ser expuestos como servicios soportando nuevos transportes.

Brinda también enrutamiento de mensajes a través de varios transportes basados en configuración de enrutamiento estático, reglas de contenido básico o balance de carga. Provee herramientas para ayudar a exponer el sistema existente como servicio web, por ejemplo, un facilitador para convertir CORBA IDL a WSDL. En adición, provee un código de conversión de solicitudes SOAP a los servicios web en peticiones IIOP para los objetos CORBA.

El ESB puede ser más que solo servir de “host” a servicios y aunque el énfasis permanece en el reuso de los sistemas existentes, un ESB puede ser usado como un potente ambiente de desarrollo de aplicaciones. Características tales como integración de bases de datos y adaptadores a aplicaciones aseguran un desarrollo rápido de aplicaciones.

Capacidades del ESB

Comunicación

Soportan enrutamiento y direccionamiento por al menos un estilo de mensajería (tales como: request/response o publish/subscribe) y al menos un protocolo de transporte que es o puede estar altamente disponible. Esto permite la sustitución de servicios y la transparencia de puntos, lo cual permite separar la vista de los servicios del consumidor de la su implementación.

Integración

Soporta varios estilos de integración o adaptadores. Hace posible la provisión de servicios basados en esta capacidad de integración y separa los aspectos técnicos de interacción de servicios y la integración de los servicios en el negocio.

Interacción de Servicios

Soporta un formato de definición de interfaz y un modelo de mensajería asociado (ej. WSDL y SOAP) para permitir la separación de los aspectos técnicos de la interacción de los servicios.

Administración y autonomía

Provee un consistente modelo de administración a través de una potentemente distribuida infraestructura, incluido control sobre nombres, enrutamiento, direccionamiento y capacidades de transformación. Esto permite la administración de los servicios en el negocio.

Capitulo 2: Panorama Internacional. Softwares usados.

La arquitectura orientada a servicios, o SOA, es una forma de ver los procesos de negocio como un conjunto de servicios vinculados. Constituye un enfoque que se vale de los estándares abiertos para que las operaciones de la empresa sean más eficientes, eficaces y fluidas. Cuando los procesos de negocios se asientan sobre una base SOA, una empresa puede lograr que sus datos y aplicaciones de software interoperen mejor entre las distintas unidades de la empresa y también con terceros externos. Este enfoque apalanca los recursos existentes a fin de contribuir a mejorar la productividad, reaccionar rápidamente a las condiciones cambiantes del mercado y aprovechar las oportunidades que se presentan.

En la actualidad existen tres principales promotores de SOA de los cuales ofrecemos las características de sus propuestas de trabajo con SOA: Microsoft, Software AG e IBM. También se expondrán los principales rasgos de la utilización de tres patrones de diseño para dicha arquitectura que hacen que estas tres compañías sobresalgan entre las demás a la hora del empleo de SOA.

Microsoft

Microsoft posee una oferta SOA que provee a los desarrolladores, arquitectos y profesionales de las tecnologías de la información las guías, tecnologías, herramientas y marco de trabajo para la construcción y mantenimiento de una solución SOA, así como los softwares que permiten a los usuarios optimizar los procesos del negocio de manera que posibilite el incremento de la productividad, disminuir los costos y promocionar la agilidad en la organización.

Describe la arquitectura SOA como un acercamiento del diseño para la organización de las tecnologías de la información en un compendio de complejos sistemas y aplicaciones distribuidas en conjunto con recursos altamente flexibles. Por lo que un proyecto de SOA bien ejecutado alinearía los recursos tecnológicos con las metas del negocio, ayudando a las empresas a construir sólidas conexiones con los clientes y suministradores, además de aumentar considerablemente la productividad de sus empleados. Básicamente, el resultado obtenido será un aumento de la agilidad en la organización.

Pero no todos los esfuerzos de orientar a servicios han sido exitosos. Los proyectos SOA tienen éxito limitado cuando se dirigen en un modelo “bottom up”, sin referencia al contexto del negocio se vuelve un proyecto sin guía ni principios organizativos, resultando en implementaciones inútiles sin relevancia en el negocio. Por su parte un modelo “top down” requiere de inversiones de tiempo tan grandes que para cuando se obtengan resultados, ya no coordinarán con las necesidades del negocio.

Microsoft por su parte aboga por lo que llama un modelo “middle out”(Microsoft 2006). En su solución, los esfuerzos de un proyecto SOA estarían dirigidos por una visión estratégica y las necesidades del negocio, que se coordinarían a través de proyectos SOA iterativos e incrementales diseñados para entregar una necesidad del negocio a la vez. Microsoft ha obtenido éxitos con su solución SOA desde 1999, cuando anunció su modelo de “Web Services” y la salida de .NET Framework y herramientas SOA. Desde entonces la solución SOA de Microsoft denominada “real world”(Microsoft 2006) ha ayudado a organizaciones a optimizar sus procesos de negocio, utilizando las herramientas, tecnologías, guías y principios de diseño SOA de Microsoft.

“Middle out”

La solución SOA de Microsoft ayuda a las organizaciones a acceder a recursos tecnológicos existentes, ensamblarlos en procesos de negocio más grandes y hace que las salidas estén disponibles a los usuarios para controlar su organización con eficacia. Esta propuesta es un hibrido de los modelos tan usados “top down” y”bottom up”. Se establecen las direcciones y prioridades a partir del negocio, entonces se comienzan a dar múltiples pasos iterativos para construir partes de las capacidades finales y con cada iteración se entrega una nueva aplicación dinámica.

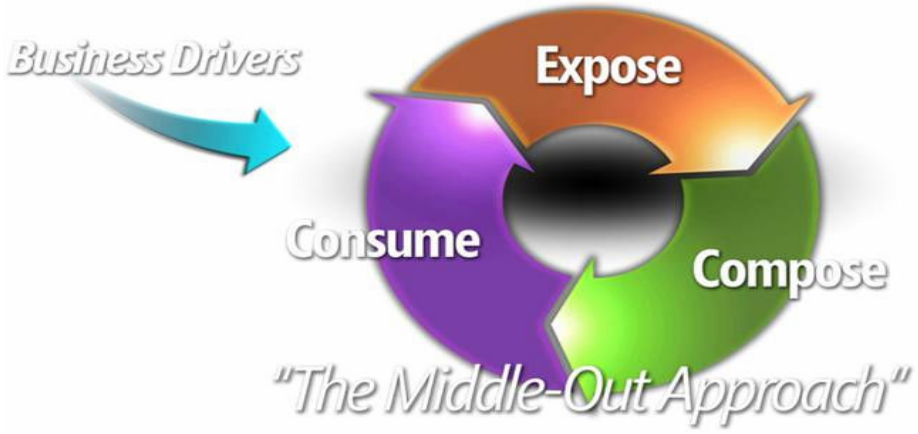

Fig. La propuesta Microsoft

Desde un punto de vista más técnico, la solución de Microsoft puede ser resumida en tres pasos: exponer, componer y consumir. Una vez que la dirección del negocio es definida, el proceso de implementar la tecnología comienza. Basados en la visión priorizada y claramente definida, cada implementación es una iteración de crear ("expose") nuevos servicios, agregar ("compose") estos servicios a procesos, y hacer que las salidas estén disponibles para el uso ("consume") de los clientes del negocio.

Expose

En la fase de exponer, los recursos tecnológicos existentes (como sistemas de herencia y aplicaciones de línea de negocios) estarán disponibles como servicios que pueden ser conectados a través de formatos estandarizados de mensajería. La forma más común de implementación de tecnologías son los servicios web. Para los bienes tecnológicos existentes que no pueden interactuar con protocolos de servicio web, la interoperabilidad se logra con el empleo de adaptadores. Mientras los desarrolladores deliberan qué exponer, sus decisiones se tomarán a partir de necesidades bien definidas y priorizadas del negocio.

Se enfoca principalmente en qué servicios crear a partir de los datos y aplicaciones subyacentes. Cada servicio estará coordinado a un proceso del negocio, tal que múltiples servicios se unifiquen para realizar un grupo de funciones del negocio. En esta fase se tiene que ver además cómo se implementan los servicios.

Compose

Una vez que son expuestos los servicios individuales, estos deben ser integrados en un proceso de negocio mayor. El objetivo de esta fase es permitir la mayor flexibilidad y agilidad del negocio, permitiendo la adición, cambio y eliminación de procesos sin entrar en conflicto con las aplicaciones y sistemas de información subyacentes.

Consume

En el paso final para la construcción de una solución SOA, son desarrolladas las aplicaciones dinámicas que consumen los procesos y servicios subyacentes. Estas aplicaciones basadas en tecnologías Web son las que conducen a la productividad del negocio. Los clientes podrán consumir los servicios compuestos a través portales Web, otros clientes, aplicaciones y dispositivos móviles.

La Plataforma SOA de Microsoft

SOA se construye en todos los aspectos de la tecnología Microsoft, desde las herramientas para desarrolladores que construyen servicios web como .NET, hasta productos servidores como BizTalk Server y el Microsoft Office SharePoint Server, además de la construcción de servicios web conectando y orquestando servicios, y la composición de aplicaciones que consumen servicios, ya sean aplicaciones Web disponibles por Internet o Intranet así como aplicaciones clientes desarrolladas con Microsoft Office o tecnologías "smart client"(Microsoft 2007).

Construyendo servicios web

Microsoft se ha comprometido a proveer todo un ecosistema para la construcción y gestión de sistemas interconectados. Ha invertido mucho en los servicios web con su plataforma de desarrollo de nueva generación Microsoft .NET

.NET Framework 3.0

Para los desarrolladores de software, .NET es el modelo de programación para construir aplicaciones en la plataforma Windows. Ofrece al sistema operativo soluciones pre-codificadas que han sido incorporadas previamente desde varias fuentes de herramientas y lenguajes de programación. Provee soporte para servicios web que permiten a los desarrolladores crear, descubrir, desplegar y consumir servicios en cualquiera de los más de 20 lenguajes que soporta .NET.

Esta versión de .NET, salida en el 2006, extiende las interfaces de aplicaciones programables de .NET Framework 2.0 con nuevas tecnologías para comunicaciones interoperables, la habilidad de modelar un amplio rango de procesos de negocio, así como gestionar las experiencias de los usuarios. Los componentes extendidos de .NET Framework 3.0 son los siguientes:

- *Windows Communication Foundation (WCF)*

La nueva generación de tecnología de servicios web de Microsoft, que facilita la conexión de sistemas y aplicaciones con la organización y sitios distribuidos por todo el mundo. Es el primer modelo de programación construido para desarrollar aplicaciones orientadas a servicios. El WCF lleva los servicios web al siguiente nivel proveyendo a los desarrolladores con un marco de trabajo altamente productivo para

la construcción de aplicaciones de servicios web que puedan interoperar a través de múltiples plataformas. Los desarrolladores se pueden enfocar en la lógica del negocio y dejarle la "plomería" subyacente al WCF. Además el WCF incluye una serie de herramientas y opciones de gestión que hace más fácil para los profesionales de las tecnologías de la información la creación, despliegue, configuración y monitoreo de los servicios web.

- *Windows Workflow Foundation*

Es un modelo de programación, motor y herramienta para la creación de aplicaciones disponibles para flujos de trabajo en Windows. Provee un marco de trabajo común para la construcción de "workflows" en aplicaciones Windows, tanto si coordinan interacciones entre softwares, personas o ambos.

Visual Studio

Es el ambiente de desarrollo profesional para construir aplicaciones en la plataforma Windows. Esta herramienta permite la creación de servicios web para aplicaciones Windows, web y Office. Además facilita a los desarrolladores la publicación y localización de nuevos servicios web, así como las pruebas que se quieran ejecutar sobre estos servicios. El Visual Studio Team System ayuda al desarrollo de la gestión a lo largo del ciclo de desarrollo de las aplicaciones, permitiendo a las organizaciones una mejor gestión de entrega y mantenimiento de las aplicaciones a través de una plataforma de control de cambios para software integrado y un sistema de reporte en tiempo real de actuación y calidad. Esto permite a los stakeholders del negocio estar actualizados en el estado del proyecto y ayuda a los equipos de software a mejorar continuamente sus procesos de desarrollo.

Integrando y orquestando procesos

BizTalk Server

Es un producto orientado a los profesionales de las tecnologías de la información y arquitectos que permite a los usuarios integrar sistemas, empleados y aliados comerciales. Con un núcleo cuya arquitectura esta basada en XML y .NET, el BizTalk soporta completamente todos los estándares abiertos en donde se construyen servicios web. Una solución de BizTalk puede consumir servicios web existentes y crear procesos de negocios como servicios web. BizTalk actúa como la capa de

gestión que orquesta servicios web, controlando el flujo y las interacciones entre ellos y agregando servicios individuales en una mayor solución compuesta.

El BizTalk Server permite además la integración de aplicaciones y sistemas que no son compatibles con servicios web. Con el uso de un rango de adaptadores, hace disponible la funcionalidad de sistemas y aplicaciones a lo largo de la organización(Microsoft 2007).

Microsoft Office SharePoint Server

Al proveer una simple y consistente experiencia de usuario a través de aplicaciones clientes conocidas, el SharePoint Server 2007 hace más fáciles y flexibles los procesos de iniciación, participación, seguimiento y reporte en los procesos de negocio centrados en el hombre. Permite a los usuarios tomar ventajas de los flujos de trabajo para automatizar y ganar visibilidad en las actividades comunes de los negocios tales como la revisión de documentos y aprobación, y el seguimiento de problemas.

El Office SharePoint Server ayuda a las organizaciones a eliminar la ineficiencia en los procesos manuales de control de información. Formularios electrónicos pueden ser utilizados para reunir información que puede ser fácilmente integrada y guardada para utilizar en procesos de flujos de trabajo o enviar a servicios web. Esta automatización ayuda a los usuarios a evitar el doble esfuerzo y los costosos errores en los procesos manuales de entrada de datos, así como asegura el acceso a información precisa en tiempo real.

Consumiendo servicios web

Microsoft ha construido soporte para los servicios web en toda su línea de productos, proveyendo a los usuarios finales de una gran conectividad para sus propios servicios(Microsoft 2007).

Windows Vista

Es la nueva salida de sistema operativo Windows, que facilita la construcción de aplicaciones más seguras, confiables y manejables. Provee de un extensivo soporte a los servicios web a través del WCF. En particular, el Windows Presentation Foundation, la nueva fundación de presentación de Windows Vista, unifica la creación,

ejecución y manipulación de documentos, multimedia e interfases de usuario en Windows, permitiendo a los desarrolladores y diseñadores crear experiencias diferenciadas y oportunidades individualizadas para sus clientes de negocio. Su cobertura de todas las formas de presentación (UI, multimedia, vectores gráficos y documentos), representa un grado de unificación totalmente nuevo en la plataforma Windows.

Office System 2007

Es la última suite productiva para trabajadores de la información, que incluye herramientas familiares de Office como el Word y el Excel. Además de estas herramientas clásicas, este Office System entrega una colección de herramientas y servicios integrados que pueden ayudar a los desarrolladores a construir soluciones de negocio utilizando servicios web, ofrece además un soporte integrado para consumir este tipo de servicios:

- InfoPath, ofrece soporte para referenciar datos a través de servicios web.
- Visio, ofrece soporte para consumir servicios web dirigidos a bases de datos, así como servicios web de bases de datos relacionales de SQL Server.

El Office System 2007 de Microsoft provee una plataforma para construir soluciones compuestas, llamada Office Business Applications (OBAs). El desarrollo de estas aplicaciones es posible gracias a los servicios del OBA, que consisten en flujos de trabajo, búsqueda, un Business Data Catalog, una nueva interfaz de usuario extensible, Microsoft Office Open XML Formats, y unos framework para sitios web y seguridad. Estos servicios pueden ser utilizados en el negocio, en comunicaciones unificadas, colaboraciones y en la gestión de contenidos de la empresa, así como otras aplicaciones del negocio de desarrolladores corporativos.

Administrando el ciclo de vida de los servicios.

Unas ves construidas y desplegadas, la infraestructura SOA y las aplicaciones dinámicas deben ser controladas a todo lo largo de su ciclo de vida. Microsoft posee una visión estratégica y una familia de productos de control de sistemas y frameworks para estas actividades(Microsoft 2007).

Dinamyc Systems Iniciative

Es la propuesta de Microsoft para entregar sistemas dinámicos auto-controlables que permitan a los equipos de IT capturar y utilizar conocimientos de sistemas de información, para diseñar sistemas más controlables y automatizar operaciones. El DSI permite a las organizaciones reducir costos y liberar tiempo para enfocarse en lo que más importante para la empresa. Actualmente Microsoft está invirtiendo fuertemente en este software para entregar mejores ofertas integradas en herramientas de desarrollo de aplicaciones, sistemas operativos, aplicaciones, hardware, tecnologías virtuales y herramientas de administración.

System Center.

Microsoft desarrolla una serie de sistemas de administración y soluciones enfocadas en proveer a los profesionales de las tecnologías de la información con las herramientas y el conocimiento para administrar sus infraestructuras. Estas soluciones se encuentran en la visión de Microsoft de integrar herramientas de administración de sistemas y otras tecnologías que ayuden a simplificar las operaciones, reducir el tiempo de resolución de problemas y mejorar las capacidades al planear la organización.

El System Center Operations Manager 2007 es una solución de administración de servicios diseñados para trabajar indistintamente con aplicaciones y softwares de Microsoft en diferentes sistemas operativos y aplicaciones para proporcionar una vista simple del ambiente IT de la organización. Esto se traduce a rapidez, ágiles respuestas a cambios y eventos que puedan impactar el curso normal del negocio o el costo de su infraestructura.

Microsoft Operations Framework

Proporciona de una guía de administración de servicios que permite a las organizaciones alcanzar la confiabilidad, disponibilidad, capacidad de soporte y de administración de Microsoft y otros productos, plataformas y tecnologías. El MOF proporciona la construcción y guía de negocios que puedan alcanzar madurez administrativa, dar prioridad a los procesos importantes, y aplicar principios y prácticas probadas para optimizar estos procesos y toda la infraestructura.

IBM

“La arquitectura SOA desempeña un papel estratégico en cualquier compañía, porque permite colaborar e innovar, identificar nuevas oportunidades de negocio y poner las necesidades de los clientes en primer plano”

Ricardo Fernández, ejecutivo del Grupo de Software IBM(Fernández 2007)

“La Administración Orientada a Servicios, o SOA, es una forma de ver los procesos del negocio como un conjunto de servicios vinculados. Constituye un enfoque que se vale de los estándares abiertos para que las operaciones de la empresa sean más eficientes, eficaces y fluidas. Cuando los procesos de negocios se asientan sobre una base SOA, una empresa puede lograr que sus datos y aplicaciones de software interoperen mejor entre las distintas unidades de la empresa y también con terceros externos. Este enfoque apalanca los recursos existentes a fin de contribuir a mejorar la productividad, reaccionar rápidamente a las condiciones cambiantes del mercado y aprovechar las oportunidades que se presentan”

José Ramón Peña, gerente de Grupo de Software de IBM(Peña 2005)

El ciclo de vida de SOA

Los clientes de IBM han manifestado que cuando piensan en SOA, lo hacen en términos de un ciclo de vida o desarrollo. Comienzan en la fase de modelado obteniendo los requerimientos del negocio y diseñando los procesos del negocio. Luego que los procesos sean optimizados, se implementan ensamblándoles nuevos o existentes servicios para formar dichos procesos del negocio. Más tarde éstos se despliegan en un ambiente de servicios integrados y altamente seguro. Después que los procesos del negocio son desplegados, se administran y monitorean desde perspectivas del negocio y de tecnologías de la información(IBM 2005). El ciclo de vida se retroalimenta de la información obtenida de esta fase de administración para mantener un continuo mejoramiento en los procesos. Bajo todas estas etapas del ciclo de vida esta la fase de gobernación y procesos, que proporciona la guía para todo el proyecto SOA.

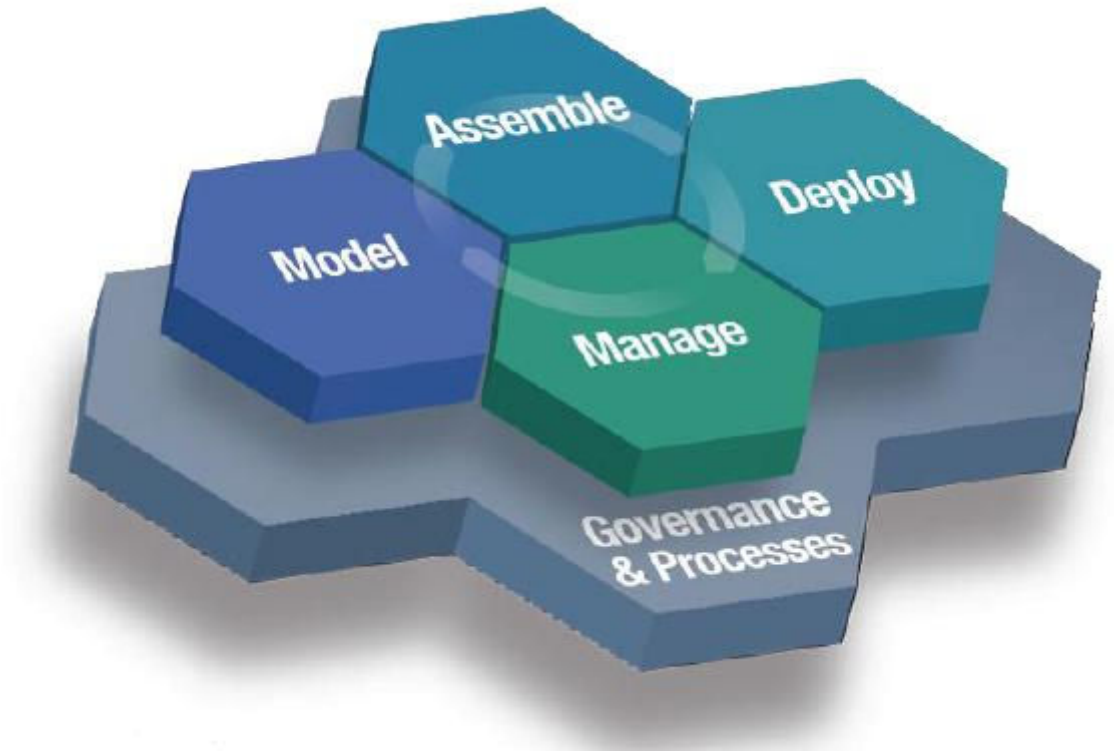

Fig. El ciclo de vida de SOA de IBM

Model

Se comienza la fase de modelado reuniendo y analizando requerimientos del negocio que entonces se utilizarán para modelar, simular y optimizar los procesos del negocio. Los procesos resultantes se utilizan para diseñar servicios de softwares asociados y niveles de servicios que soporten estos procesos. Durante esta fase, se usa un modelo para establecer un entendimiento común entre el negocio y las IT de los procesos del negocio, los objetivos y las salidas, así como para asegurarse que la aplicación resultante cumpla con los requerimientos del negocio definidos. Este modelo provee además la línea base desde donde se medirá el rendimiento del negocio.

Assemble

Durante la fase de ensamblar, se crean los servicios a partir bienes existentes, como sistemas financieros, aplicaciones y otras soluciones que corran en el negocio. En muchos casos, una librería de servicios existentes se puede utilizar para encontrar servicios que ya existan en la organización. Si no existe funcionalidad, se puede crear y probar un servicio que entregue la funcionalidad requerida para un proceso particular del negocio. Luego que los servicios requeridos estén disponibles, son orquestados para implementar un proceso del negocio.

Deploy

En esta fase se configura y escala el ambiente de ejecución donde se lograrán los niveles de servicio requeridos por los procesos del negocio. Después que esté configurado un proceso, éste se puede desplegar a un ambiente de servicios robusto, escalable y altamente seguro. Este ambiente se optimiza para que pueda ejecutar confiadamente misiones críticas de los procesos del negocio a la vez que provee flexibilidad para las actualizaciones dinámicas en respuesta al cambio de los requerimientos del negocio. Este acercamiento o solución orientada a servicios reduce el costo y la complejidad asociados al mantenimiento de numerosas integraciones point-to-point.

Manage

La fase de administración implica establecer y mantener la disponibilidad y tiempo de respuesta de los servicios, así como administrar los bienes de servicios subyacentes. Se pueden monitorear los indicadores claves de rendimiento KPI ("key performance indicators") en tiempo real para obtener la información necesaria para prevenir, aislar, diagnosticar y arreglar problemas. Comprender el rendimiento en tiempo real de los procesos del negocio permite la retroalimentación vital para el modelo de procesos del negocio, de manera que se establezca una mejora continua. Esta fase incluye además la administración y mantenimiento del control de versiones sobre los servicios que conforman los procesos del negocio.

Governance and processes

Esta fase es crítica para el éxito de cualquier proyecto SOA. Para asegurar este éxito, se puede crear un centro de excelencia dentro del negocio para implementar las políticas de gobierno y seguir comprobados estándares internacionales de control de información y tecnologías relacionadas. Implementar fuertes políticas do gobierno puede dar resultado a un exitoso proyecto de SOA, incluso tiene el potencial de obtener mayores ganancias e incrementar los valores de las acciones empresariales.

Fundación IBM SOA

IBM SOA Foundation(IBM 2005) es un grupo de software integrado basado es estándares abiertos, guías y patrones, que se ha diseñado para proporcionar lo necesario para comenzar y desarrollar un proyecto SOA. Todo el software que incluye el IBM SOA Foundation ha sido escogido cuidadosamente de los softwares punteros

de IBM que se aplican a cada fase del ciclo de vida SOA. IBM SOA Foundation es interoperable, permitiendo la selección de componentes mediante la base de build-as-you-go, es escalable, ya que se puede comenzar con pequeños proyectos e ir aumentando la complejidad tan rápido como el negocio lo requiera y además proporciona soporte al negocio y a los estándares de IT para facilitar una mayor interoperabilidad y portabilidad entre las aplicaciones(IBM 2007).

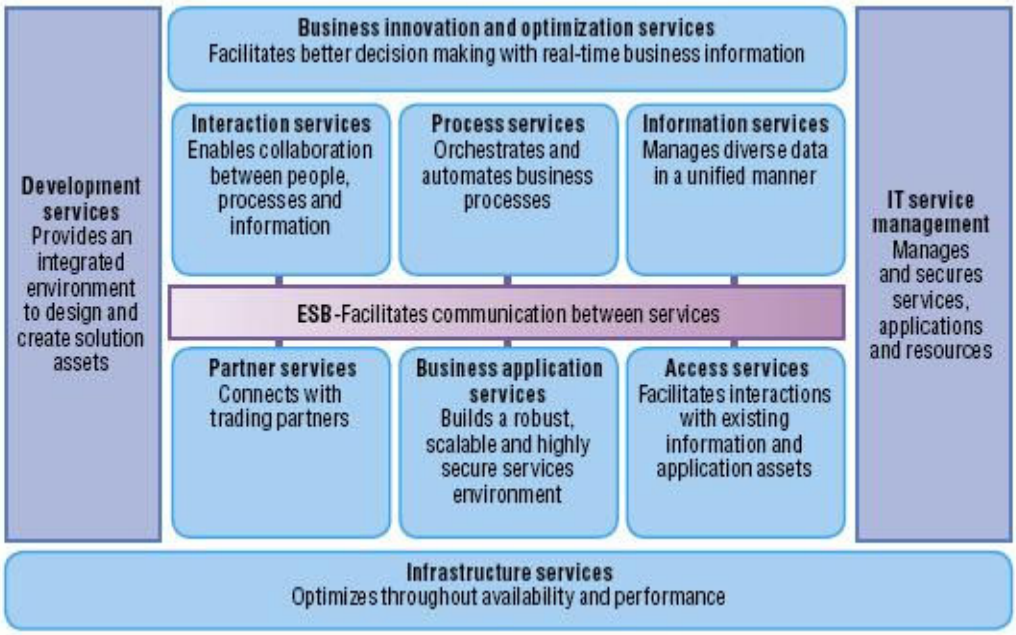

Fig: Arquitectura IBM SOA Foundation

IBM SOA Foundation esta basada en una arquitectura que define los servicios requeridos en cada etapa del ciclo de vida SOA.

Model

Desarrollando servicios. IBM SOA Foundation ofrece una plataforma de desarrollo basada en roles y unificada, que permite a cada miembro del equipo de desarrollo completar eficientemente las tareas. Durante la fase de modelado las herramientas están orientadas para que los analistas del negocio evalúen requerimientos del negocio, y modelen y simulen procesos del negocio.

Los siguientes productos están orientados para esta fase:

- IBM WebSphere Business modeler
- IBM Racional Software Architect

Assemble

Desarrollando servicios. En esta fase las herramientas para los arquitectos de software les permiten modelar datos e interacciones del sistema. Los especialistas de integración pueden utilizar herramientas para configurar las interacciones entre los procesos del negocio. IBM SOA Foundation proporciona las herramientas para crear nuevos servicios, permitiendo a los desarrolladores crear nuevas lógicas del negocio.

Los siguientes productos están orientados para esta fase:

- IBM WebSphere Integration Developer
- IBM Racional Application Developer

Deploy

Enterprise Service Bus. El ESB permite toda la capacidad de interconexión requerida para utilizar los servicios implementados a lo largo de la arquitectura(IBM 2007).

- Servicios de interacción: estos servicios proporcionan la entrega de funciones y datos a los usuarios según sus preferencias específicas.
- Servicios de procesos: proporcionan el control para administrar el flujo y la interacción de múltiples servicios de modo que implementen procesos del negocio.
- Servicios de información: para responder y transformar datos.
- Servicios de acceso: proporcionan los puentes entre aplicaciones de núcleo, aplicaciones pre-empaquetadas, bases de datos y el ESB, para incorporar servicios de aplicaciones existentes.
- Servicios de Partner: estos servicios proporcionan los documentos y protocolos para los procesos del negocio que involucren interacciones con suministradores y otros socios del negocio.
- Servicios de aplicaciones del negocio: son los servicios de ejecución que incluyen los componentes de aplicaciones necesarios para el sistema integrado.
- Servicios de infraestructura: servicios para optimizar la disponibilidad y ejecución.

IBM SOA Foundation proporciona los siguientes productos para esta fase del ciclo de vida SOA:

- IBM WebSphere Process Server
- IBM WebSphere ESB
- IBM WebSphere Message Broker
- IBM WebSphere Partner Gateway
- IBM WebSphere Adapters
- IBM WebSphere Portal
- IBM WebSphere Everyplace Deployment
- IBM WebSphere Collaboration Services
- IBM WebSphere Information Integrator
- IBM WebSphere Application Server
- IBM WebSphere Extended Deployment

Manage

Administración de servicios. Estos servicios permiten la escalabilidad y ejecución.
Los siguientes productos están orientados para esta fase:

- IBM WebSphere Business Monitor
- IBM Tivoli Composite Application Manager
- IBM Tivoli Identity Manager
- IBM Tivoli Acces Manager

Software AG

Software AG es una compañía alemana líder mundial en soluciones de TI y con gran renombre por sus confiables y eficaces soluciones de software para misión crítica. Ofrece además el poder para crear, administrar y gobernar de manera rápida y rentable nuevos procesos de negocio dentro de una SOA, brindando el control que se necesita para alcanzar la visión de negocios que se desea(AG 2007).

Navegador SOA

Crossvision incluye seis potentes componentes que le brindan el control que necesita para lograr su visión de negocio.

Fig: crossvision. Propuesta de Software AG.

Componentes

crossvision está formado por seis componentes modulares basados en estándares abiertos de la industria.

Componentes	Beneficios
Crossvision Application Composer	
Desarrolla rápidamente nuevas aplicaciones empresariales a partir de los sistemas actuales	● Crea visualmente nuevas soluciones empresariales sin exigir conocimientos de programación especializados. ● Crea Rich Internet Applications (RIA) aplicaciones optimizadas para obtener el máximo provecho del navegador a partir de los actuales

	servicios y componentes. ● Acelera y simplifica el diseño e implantación de las aplicaciones.
Crossvision Business Process Manager	
Coordina el flujo de los procesos empresariales de toda la organización	● Modela, automatiza y optimiza los procesos empresariales. ● Evalúa los resultados. ● Mejora la productividad al incrementar la agilidad. ● Aumenta la utilización de los recursos y fuerza el seguimiento de los procesos.
Crossvision Service Orchestrator	
Crea nuevos servicios empresariales a partir de las actuales funciones y recursos técnicos	● Crea servicios empresariales de alta rentabilidad a partir de estándares abiertos. ● Transforma los servicios y mensajes actuales mediante su composición y coordinación. ● Aprovecha los avanzados mecanismos de seguridad y estabilidad.
Crossvision Information Integrator	
Combina datos procedentes de diferentes sistemas en una única y sencilla vista	● Gestiona con eficacia la información ● Modela vistas de la empresa ● Incorpora la lógica empresarial ● Publica modelos de información basados en un avanzado enfoque semántico de la web

Crossvision Legacy Integrator	
Integra los activos de la empresa y crea nuevas funciones y recursos	● Aprovecha los actuales sistemas como nuevos servicios ● Evita la indeseable destrucción o sustitución ● Encapsula las funciones, amplía la efectividad y aprovecha al máximo todos los activos de TI
CentraSite	
Gestiona y dirige el entorno SOA para conseguir la máxima accesibilidad y transparencia	● Gestiona todo el ciclo vital de la SOA basado en estándares abiertos. ● Se beneficia de una mayor transparencia y colaboración, una utilización eficiente de los servicios, un proceso de cambios sin problemas y una óptima gestión. ● Aprovecha las ventajas de una estructura troncal abierta y de gran conectividad que integra todas las herramientas SOA y les permite interactuar entre sí.

crossvision Application Composer

Software AG ofrece todo lo que se necesita para diseñar interfaces SOA fáciles de utilizar tanto para conseguir nuevas oportunidades de negocio como para acelerar la salida de nuevos productos. Para resolver todos los problemas actuales, como el de aplicaciones empresariales híbridas para la web, las cuales son nuevas y bastante complejas, es necesario adoptar entornos colaborativos dotados de interfaces adecuadas entre las personas y los procesos de su SOA. La propuesta de Software

AG es entonces crossvision Application Composer el cual parece ayudar a hacer frente a esta problemática.

Con crossvision Application Composer se pueden transformar los actuales servicios de negocios en nuevas soluciones de usuario final a partir de un planteamiento declarativo y agnóstico ante las tecnologías. Es el primer producto orientado al desarrollo de interfaces de usuario para entornos SOA y aplicaciones web híbridas. A la vez que evita la complejidad inherente a la creación de interfaces de usuario, proporciona un marco que permite entornos muy colaborativos que aprovechan las prestaciones de Web 2.0(AG 2007).

crossvision Application Composer ha ayudado a numerosos y conocidos clientes de todo el mundo a llevar a cabo exitosos proyectos de misión crítica, como es el caso de una interfaz de usuario entre el sistema de recursos humanos y la central de compras para Nissan Motor Co. (Australia), un nuevo sistema de gestión de tiempos para FCCJ (EE.UU.), así como la modernización e integración completa de un sistema de punto de venta, Intranet y almacén de datos centralizado para Apollo Optik (Alemania).

Ventajas

- Aumentar al máximo la productividad de los usuarios finales con interfaces Web 2.0
- .Facilitar el desarrollo rápido y sencillo de aplicaciones web híbridas
- Utilizar una estrategia de distribución flexible para conseguir de forma equilibrada desarrollos asequibles, sofisticados y accesibles
- .Facilitar un diseño rápido e intuitivo
- .Acelerar los actuales ciclos de distribución de productos y servicios
- .Lanzar con mayor rapidez nuevos productos y servicios.

La propuesta de Software AG es robusta:

- Soportando e integrando procesos de negocio flexibles a través de la organización
- .Re-utilizando y optimizando sus sistemas heredados haciéndolos disponibles para su SOA
- .Gestionando y gobernando su visión SOA.

crossvision Application Designer

Crossvision Application Designer provee una solución para desarrollar aplicaciones RIA en muy poco tiempo. Permite al "web browser" navegar como una interfaz de usuario grafica tradicional con grandes funcionalidades, tales como clientes de correo electrónico, con inclusiones de menús de click derecho y teclas de acceso rápido. Se puede con este componente tener buen provecho de los beneficios de AJAX (Asynchronous JavaScript XML Communication) sin la necesidad de meterse en el código(AG 2007).

Ventajas

- Incrementa la interactividad de las aplicaciones de negocio basadas en web para maximizar la eficiencia y satisfacción del usuario.
- Presenta un plano e interactivo programa experimentado en principios de desarrollo easy-to-use (fácil de usar).
- Elimina la complejidad de desarrollo de AJAX a través de un diseño basado en J2EE y entorno de ejecución.
- Entrega rápida de nuevas aplicaciones de negocio.
- Hace más flexible la estrategia de despliegue con web o clientes pesados para balancear la accesibilidad.
- Actualiza automáticamente a través de un servidor GUI integrado.

crossvision Business Process Manager

crossvision Business Process Manager proporciona agilidad para definir, ejecutar y optimizar los procesos empresariales así como para evaluar los resultados. Brinda la posibilidad de desmarcarse de la competencia con la integración de personas, procesos y datos para conferir el máximo dinamismo al negocio, incrementar la productividad y mejorar los márgenes. crossvision Business Process Manager proporciona la agilidad necesaria para alcanzar estos objetivos porque permite coordinar y optimizar todos los procesos empresariales de la organización.

Las organizaciones que alinean los procesos y sistemas de TI con su estrategia empresarial consiguen aplicaciones de gran calidad, una ventaja competitiva

fundamental en un entorno en continua evolución. crossvision Business Process Manager proporciona la base idónea para esta integración(AG 2007):

- Adapta rápidamente los procesos a las nuevas condiciones del mercado y al cumplimiento de las normas.
- Mejora la productividad al incrementar la agilidad.
- Automatiza los procesos manuales.
- Reduce el tiempo de respuesta a los clientes.
- Aumenta la utilización de los recursos y fuerza el seguimiento de los procesos.
- Identifica cuellos de botella y aumenta al máximo la eficacia.
- Acelera la salida al mercado de nuevos productos y servicios.

Agilidad, eficacia y flexibilidad son los principales ingredientes para potenciar el rendimiento de los recursos. crossvision Business Process Manager ayuda a gestionar mejor las tareas de toda la organización al proponerle un enfoque basado en procesos y centrado en la información para:

- *Definir procesos empresariales:* incluidas la documentación de tareas y la identificación de roles, responsabilidades y recursos.
- *Ejecutar procesos empresariales*: que facilitan la interacción entre las personas, datos y sistemas para garantizar que se respetan las instrucciones y se registran las acciones a la vez que se integra con los sistemas subyacentes para intercambiar datos, mejorar la precisión y racionalizar las operaciones.
- *Optimizar procesos empresariales:* mediante el control y análisis de actividades, duración de procesos y utilización de recursos para identificar las oportunidades de mejora, simular situaciones hipotéticas e implantar cambios.
- *Alinear a las personas y sistemas de TI con la estrategia empresarial para conseguir procesos de calidad y una ventaja competitiva.*

crossvision Information Integrator

crossvision Information Integrator puede gestionar y fusionar datos, definir dependencias y publicar servicios empresariales para suministrar la información que se necesite mediante tecnologías semánticas avanzadas.

La mayoría de las organizaciones tienen los datos de la empresa distribuidos por diferentes sistemas y bases de datos compartimentadas. Los usuarios de la empresa necesitan acceder a diferentes fuentes de datos siguiendo las instrucciones de la organización para conseguir una vista relevante y clara de la información que les permita tomar decisiones efectivas para el rápido desarrollo de los procesos empresariales. crossvision Information Integrator puede describir y agregar de forma dinámica los datos en una única vista reutilizable en función de las necesidades de los usuarios y de la semántica utilizada(AG 2007).

crossvision Information Integrator obtiene la información de diferentes fuentes y la presenta al usuario de forma elocuente, precisa y relevante para permitirle:

- Gestionar con eficacia la información.
- Modelar vistas del negocio.
- Incorporar la lógica empresarial.
- Publicar modelos de información basados en un avanzado enfoque semántico.
- Acceder a información de diferentes bases de datos desde una sola vista.
- Tomar decisiones mejor fundadas y acelerar los procesos empresariales.

crossvision Information Integrator combina un enfoque único basado en modelos en el que los conceptos empresariales están visualmente asignados a las fuentes de datos, haciendo especial hincapié en las directrices del negocio. Es una forma de ayudar a definir las relaciones y conciliar las diferencias entre conceptos empresariales.

Las vistas individuales se generan a partir de modelos, y con terminología relevante para los usuarios de la empresa, se presentan como servicios web que pueden aprovechar los portales, las aplicaciones compuestas, los procesos empresariales u otros componentes SOA. crossvision Information Integrator fomenta de este modo un enfoque SOA para la integración de datos, ya que aprovecha tanto los modelos como las vistas independientes.

crossvision Legacy Integrator

Integra los activos de la empresa y crea nuevas funciones y recursos. Abre los actuales sistemas como nuevos servicios para evitar la indeseable sustitución o

destrucción de activos Puede encapsular las funciones, ampliar la efectividad y aprovechar al máximo todos sus activos de TI.

Gracias a crossvision Legacy Integrator se pueden aprovechar el potencial y enorme valor de los sistemas de la empresa para conservar la eficiencia operativa y ventaja competitiva de la organización. Al considerar que todos los sistemas, nuevos y antiguos, son miembros de SOA de primera clase, garantiza el máximo aprovechamiento y flexibilidad(AG 2007).

crossvision Legacy Integrator está especialmente diseñado para desbloquear los datos y funciones de los sistemas heredados y preservar así la fiabilidad, seguridad y rendimiento sin recurrir a costosas inversiones. crossvision Legacy Integrator permite:

- Adaptar los esfuerzos de integración a la infraestructura de TI y al ritmo y necesidades particulares.
- Conseguir con mayor rapidez los objetivos de negocio en pocos días mediante pruebas de concepto.
- Aumentar la eficacia, competitividad y productividad.
- Proporcionar potentes servicios a clientes remotos a través de la Web.
- Preservar y ampliar el valor de las aplicaciones heredadas.

crossvision Legacy Integrator se erige sobre la probada experiencia de Software AG fraguada con más de un millar de proyectos de integración que han proporcionado soluciones de alto rendimiento y gran solidez a cientos de organizaciones de todo el mundo. El desarrollo sin esfuerzo de nuevos servicios a partir del aprovechamiento de los actuales activos favorece la creación de soluciones empresariales a tiempo y dentro del presupuesto. La gestión de estos servicios desde el repositorio/registro de CentraSite garantiza la máxima reutilización de recursos y la óptima gestión dentro de su Arquitectura orientada a servicios.

crossvision Master Data Manager

Actualmente, la información de las empresas se encuentra repartida entre varios sistemas heterogéneos. Dichos sistemas almacenan datos maestros, que normalmente incluyen información relativa a los productos, los empleados, datos de los clientes u otros datos de referencia críticos para el negocio. Como los

mencionados datos de referencia se encuentran almacenados en distintos sistemas y, normalmente, en diferentes formatos, resulta imposible disponer de un acceso unificado a toda esa información crítica para la empresa. crossvision Master Data Manager unifica, gestiona y proporciona datos de referencia coherentes y exhaustivos a lo largo de toda la empresa.

crossvision Master Data Manager permite administrar todo el ciclo de vida de los datos maestros, independientemente del tipo o ubicación de los mismos. Ayuda a las empresas a mantener una visión única del negocio con respecto a los datos de referencia. crossvision Master Data Manager consolida los datos procedentes de distintas fuentes en un modelo de datos unificado. Posteriormente, dicha información se dispone como un servicio, que puede ser utilizado prácticamente por cualquier sistema o aplicación SOA.(AG 2007)

Ventajas

- Unificación y consolidación de datos maestros redundantes – repartidos a lo largo de los sistemas informáticos – dentro de un repositorio centralizado de datos maestros.
- Implementación de un sistema único de gestión de datos maestros para todos los datos maestros de la empresa.
- Los modelos de datos abiertos garantizan la adaptación a cualquier requisito empresarial.
- Seguridad para los datos de referencia mediante la delegación de derechos vía herencia.
- Capacitación SOA mediante la exposición de datos maestros consolidados a través de servicios web.
- La validación y verificación de los cambios en los datos maestros se rigen mediante un sólido soporte del ciclo de vida dentro de los procesos empresariales.
- Habilitación de usuarios de negocio para la administración de datos maestros con una interfaz gráfica de usuario fácil de usar.

crossvision Master Data Manager permite a las empresas organizar todos sus datos maestros dispersos, independientemente de su ubicación o sistema, y proporciona un mecanismo de soporte para todo el ciclo de vida de cualquier tipo de dato maestro.

crossvision Service Orchestrator

crossvision Service Orchestrator permite crear servicios empresariales de gran valor. Gracias a las prestaciones avanzadas de Enterprise Service Bus, se pueden componer y orquestar los actuales servicios y mensajes de la empresa para formar nuevos servicios empresariales de gran valor. En los últimos diez años, la integración de sistemas de negocio ha emergido como un medio clave para las empresas de competir con más eficacia, mejorar la productividad y generar un continuo rendimiento de las inversiones en TI. El modelo de integración predominante actualmente es SOA, una arquitectura orientada a servicios que encapsula las funciones de la empresa.

La verdadera fuerza de SOA es crear nuevas aplicaciones y servicios sin necesidad de reinstalar las funciones y recursos existentes. crossvision Service Orchestrator se ha diseñado especialmente para SOA al ofrecer un Enterprise Service Bus (ESB) específico para las empresas y basado en estándares abiertos. Siguiendo los principios de ESB, crossvision Service Orchestrator le permite aprovechar los activos de TI y organizarlos en torno a nuevos y potentes servicios empresariales que proporcionan la adecuada granularidad para responder de forma óptima a las necesidades del negocio(AG 2007):

- Orquestación: creación y reorganización de servicios basados en el potente estándar BPEL para la orquestación de procesos.
- Composición: composición de nuevos servicios empresariales de alto valor a partir de los actuales sistemas basados en enrutamiento por contenidos, extenso soporte de transformaciones y amplia conectividad.
- Seguridad: encriptación y firma conforme a las especificaciones de seguridad XML.
- Auditoria: conservación de registros e imágenes con fines informativos y de prevención de rechazos.
- Escalabilidad: expansión y equilibrio dinámico de la carga entre varios equipos.

En conexión con CentraSite, repositorio SOA central y estructura troncal de gran conectividad de la suite crossvision, se podrá potenciar con facilidad las funciones y servicios en una solución SOA procedentes de distintas fuentes.

CentraSite

Gestiona todo el ciclo vital de SOA basado en estándares abiertos. La organización se beneficia de mayor transparencia y colaboración, una eficiente utilización de los servicios, un proceso de cambios sin problemas y una óptima gestión. Se obtiene una perspectiva más amplia que facilita el proceso de gestión y el cumplimiento de las medidas adoptadas. Esto ayuda a implantar SOA de forma satisfactoria y, en consecuencia, a acortar los tiempos de salida al mercado. Proporciona un control completo sobre servicios acoplados de forma flexible y garantiza así la gestión y transparencia de iniciativas SOA.

CentraSite es un repositorio SOA de próxima generación desarrollado conjuntamente por Fujitsu y Software AG que crea una base abierta para la gestión de proyectos SOA. Ofrece una incomparable flexibilidad, visibilidad y control de SOA y crea un entorno de colaboración más fértil entre los diferentes equipos de la organización. Es extremadamente flexible y capaz de gestionar metadatos de diferentes activos SOA. A los analistas, se les provee de información relevante sobre los procesos y servicios empresariales, como directrices y modelos de información semántica. Los diseñadores y desarrolladores pueden manejar servicios así como descripciones y asignaciones de datos(AG 2007).

Al proporcionar la máxima transparencia sobre los activos SOA, CentraSite ayuda a:

- Gestionar todo el ciclo vital de SOA basado en estándares abiertos.
- Potenciar el valor de los sistemas de TI al promocionar la reutilización de los activos nuevos y antiguos.
- Conseguir la fiabilidad que otorgan los análisis de impacto previos a la aplicación de los cambios.
- Obtener las ventajas de una mayor transparencia y una mejor colaboración.

¿Qué hace SOA?

Estrictamente hablando, SOA está hecha por desarrolladores y arquitectos de soluciones, sin embargo, los proveedores de las soluciones orientadas a servicios amplían el rango de roles y es crítico que sus intereses no sean solo tomados en cuenta sino que activamente manejan el diseño de las soluciones SOA.

Comenzando con dichos intereses, el analista de negocio es el encargado de llevar a cabo las inversiones relacionadas con las TIC específicamente o mayormente en la línea de estrategia de negocio. Para los desarrolladores, esto significa que la solución SOA debe mapear la fuente del sistema de información de negocio, personal, socios de comercio, en una vista unificada y comprensiva tal como analista de negocio haya tenido el grado de entendimiento sobre el costo y los beneficios de varias inversiones(Chapell 2007).

Lo que no es SOA

SOA no es un producto disponible a la venta, es una filosofía de diseño que informa como se deberá construir el resultado. SOA no es el objetivo, es un medio para lograr un resultado.

No requiere un nuevo proceso o una tecnología nueva y exclusiva, al contrario, las soluciones SOA deben ser incrementales y construidas bajo las financiaciones actuales del negocio.

SOA a menudo es malinterpretada y confundida con los “Web Services”, aunque es cierto que SOA es más fácil bajo la adopción de estándares y protocolos basados en servicios web, son marcadamente distintos. SOA es un acercamiento al diseño de sistemas, dirige como los recursos tecnológicos serán integrados y que servicios serán expuestos. A su vez los servicios web son una metodología de implementación que usa estándares específicos y protocolos de lenguaje para ejecutar la solución SOA.

Capitulo 3: Pautas para la integración de aplicaciones

En este capítulo se propone un modelo SOA que se ajuste a las características de los sistemas que se desarrollan en la fábrica de portales de la Facultad 10 de la Universidad de las Ciencias Informáticas. Pero antes de proceder a las pautas de la implementación de la propuesta se explicará qué son los servicios web para comprender como funcionará la solución SOA propuesta.

Servicios web

Existen múltiples definiciones sobre lo que son los servicios web, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Una posible sería hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la web(W3C 2006). Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

Un servicio web (en inglés “web service”) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes y ejecutadas sobre cualquier plataforma pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet(Shodjai 2006). La interoperabilidad se consigue mediante la adopción de estándares abiertos.

Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.(WS-I 2006)

Estándares

WSDL

Son las siglas de “Web Services Description Language”, un formato XML que se utiliza para describir servicios web

WSDL es un formato XML que describe los servicios de red como un conjunto de puntos finales que procesan mensajes contenedores de información orientada tanto a documentos como a procedimientos(Wolter 2001). Las operaciones y los mensajes se describen de forma abstracta y después se enlazan a un protocolo de red y a un formato de mensaje concreto para definir un punto final de red. Los puntos finales concretos relacionados se combinan en puntos finales abstractos (servicios). WSDL es extensible, lo que permite la descripción de puntos finales de red y sus mensajes, independientemente de los formatos de los mensajes o protocolos de red utilizados para comunicarse.

Dado que los protocolos de comunicaciones y los formatos de mensajes están estandarizados en la comunidad del Web, cada día aumenta la posibilidad e importancia de describir las comunicaciones de forma estructurada. WSDL afronta esta necesidad definiendo una gramática XML que describe los servicios de red como colecciones de puntos finales de comunicación capaces de intercambiar mensajes. Las definiciones de servicio de WSDL proporcionan documentación para sistemas distribuidos y sirven como fórmula para automatizar los detalles que toman parte en la comunicación entre aplicaciones.(Barco 2006)

Los documentos WSDL definen los servicios como colecciones de puntos finales de red o puertos. En WSDL, la definición abstracta de puntos finales y de mensajes se separa de la instalación concreta de red o de los enlaces del formato de datos(Erik Christensen 2000). Esto permite la reutilización de definiciones abstractas: mensajes, que son descripciones abstractas de los datos que se están intercambiando y tipos de puertos, que son colecciones abstractas de operaciones. Las especificaciones concretas del protocolo y del formato de datos para un tipo de puerto determinado constituyen un enlace reutilizable. Un puerto se define por la asociación de una dirección de red y un enlace reutilizable; una colección de puertos define un servicio. Por esta razón, un documento WSDL utiliza los siguientes elementos en la definición de servicios de red[1]:

- Types: contenedor de definiciones del tipo de datos que utiliza algún sistema de tipos (por ejemplo XSD).
- Message: definición abstracta y escrita de los datos que se están comunicando.

[1] Anexo1: *helloservice.wsdl*

- Operation: descripción abstracta de una acción admitida por el servicio.
- Port Type: conjunto abstracto de operaciones admitidas por uno o más puntos finales.
- Binding: especificación del protocolo y del formato de datos para un tipo de puerto determinado.
- Port: punto final único que se define como la combinación de un enlace y una dirección de red.
- Service: colección de puntos finales relacionados.

SOAP

SOAP es el protocolo de comunicaciones para los servicios web XML.
Protocolo Simple de Acceso a Objetos ("Simple Object Access Protocol"). Se trata de un protocolo basado en XML, que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de HTTP, SMTP, entre otros(Nilo Mitra 2006). SOAP especifica el formato de los mensajes. El mensaje SOAP está compuesto por un "*envelope"* (sobre), cuya estructura está formada por los elementos "*header"* (cabecera) y "*body"* (cuerpo).(Wolter 2001)

XML

Siglas en inglés de "eXtensible Markup Language" (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el "World Wide Web Consortium" (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos. XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas.(XML 2007)

Se define servicio web XML como un servicio software expuesto en la Web mediante SOAP, descrito con un archivo WSDL y registrado en UDDI. Una de las ventajas principales de la arquitectura de servicios web XML es que permite a los programas escritos en diferentes lenguajes sobre diferentes plataformas comunicarse entre sí de un modo basado en estándares.

Los servicios web XML son los bloques de construcción básicos en la transición al proceso distribuido en Internet. Los estándares abiertos y el foco en la comunicación y

colaboración entre las personas y aplicaciones han creado un entorno donde los servicios web XML se están convirtiendo en la plataforma para la integración de aplicaciones. Las aplicaciones se construyen utilizando múltiples servicios web XML desde diversas fuentes que trabajan conjuntamente con independencia de dónde residen o cómo hayan sido implementadas(Wolter 2001).

- Los servicios web XML exponen funcionalidad útil a los usuarios mediante un protocolo web estándar. En la mayoría de casos, el protocolo utilizado es Simple Object Access Protocol (SOAP).
- Los servicios web XML proporcionan un modo de describir sus interfaces con suficiente detalle para permitir a un usuario construir una aplicación cliente para hablar con ellos. Esta descripción se proporciona generalmente en un documento XML que responde al nombre de documento “Web Services Description Language” (WSDL).
- Los servicios web XML se registran de modo que los potenciales usuarios puedan encontrarlos. Esto se realiza mediante “Universal Discovery Description and Integration” (UDDI).

¿Qué tipos de servicios web existen?

Esta pregunta se la hace todo desarrollador a la hora de enfrentarse a un problema de este tipo. Existen varias clasificaciones dependiendo de su autor. Existen básicamente tres tipos de servicios, divididos sobre la base de sus funcionalidades(Microsoft 2007):

Servicios controladores: Son los encargados de recibir las peticiones de los clientes y realizar las llamadas necesarias a otros servicios (en la secuencia adecuada) para devolver una respuesta. Es decir, son los servicios encargados de coordinar al resto de servicios. Este tipo de servicio representa a los procesos de negocio que se desea implementar, ya que un proceso de negocio no es más que un conjunto de tareas ejecutadas en una determinada secuencia para obtener un objetivo.

Servicios de negocio: Son los servicios que representan una tarea de negocio, y que forman parte de un proceso de negocio. Este tipo de servicios suelen ser poco reutilizables porque están orientados a resolver una tarea muy puntual.

Servicios de utilidad: Son aquellos servicios que se caracterizan por representar una tarea altamente reutilizable. Existen dos tipos, los servicios orientados al negocio que representan una tarea de negocio altamente reutilizable entre aplicaciones y los servicios tecnológicos encargados de encapsular una determinada tecnología y por tanto altamente reutilizables (ej: servicio de acceso a bases de datos relacionales).

Principios de la orientación a servicios

No existe una definición estándar de cuales son los Principios de la Orientación a Servicios, por lo tanto, lo único que se puede proporcionar es un conjunto de Principios que estén muy asociados con la Orientación a Servicios.
Cuando se desarrollan aplicaciones SOA es muy útil y necesario tener en cuenta siempre estos principios, ya que van a dar las pautas necesarias para tomar ciertas decisiones de diseño complejas(Barco 2006).

Estos Principios según Thomas Erl son:

- Los Servicios deben ser reusables: Todo servicio debe ser diseñado y construido pensando en su reutilización dentro de la misma aplicación, dentro del dominio de aplicaciones de la empresa o incluso dentro del dominio público para su uso masivo.

- Los Servicios deben proporcionar un contrato formal: Todo servicio desarrollado, debe proporcionar un contrato en el cual figuren: el nombre del servicio, su forma de acceso, las funcionalidades que ofrece, los datos de entrada de cada una de las funcionalidades y los datos de salida. De esta manera, todo consumidor del servicio, accederá a este mediante el contrato, logrando así la independencia entre el consumidor y la implementación del propio servicio. En el caso de los servicios web, esto se logrará mediante la definición de interfaces con WSDL.

- Los Servicios deben tener bajo acoplamiento: Es decir, que los servicios tienen que ser independientes los unos de los otros. Para lograr ese bajo acoplamiento, lo que se hará es que cada vez que se vaya a ejecutar un servicio, se accederá a él a través del contrato, logrando así la independencia entre el servicio que se va a ejecutar y el que lo llama. Si conseguimos este bajo acoplamiento, entonces los servicios podrán ser totalmente reutilizables.

- Los Servicios deben permitir la composición: Todo servicio debe ser construido de tal manera que pueda ser utilizado para construir servicios genéricos de más alto nivel, el cual estará compuesto de servicios de más bajo nivel. En el caso de los servicios web, esto se logrará mediante el uso de los protocolos para orquestación (WS-BPEL) y coreografía (WS-CDL).

- Los Servicios deben de ser autónomos: Todo Servicio debe tener su propio entorno de ejecución. De esta manera el servicio es totalmente independiente y se puede asegurar que así podrá ser reutilizable desde el punto de vista de la plataforma de ejecución.

- Los Servicios no deben tener estado: Un servicio no debe guardar ningún tipo de información. Esto es así porque una aplicación está formada por un conjunto de servicios, lo que implica que si un servicio almacena algún tipo de información, se pueden producir problemas de inconsistencia de datos. La solución es que un servicio sólo contenga lógica, y que toda información esté almacenada en algún sistema de información sea del tipo que sea.
- Los Servicios deben poder ser descubiertos: Todo servicio debe poder ser descubierto de alguna forma para que pueda ser utilizado, consiguiendo así evitar la creación accidental de servicios que proporcionen las mismas funcionalidades. En el caso de los servicios web, el descubrimiento se logrará publicando los interfaces de los servicios en registros UDDI

Una característica muy importante de los principios de la orientación a servicios, es que todos ellos se interrelacionan. Exponer procesos de negocio como servicios es la clave a la flexibilidad de la arquitectura. Esto permite que otras piezas de funcionalidad (incluso también implementadas como servicios) hagan uso de otros servicios de manera natural, sin importar su ubicación física. Así un sistema evoluciona con la adición de nuevos servicios y su mejoramiento. Donde cada servicio evoluciona de una manera independiente.

La arquitectura orientada a servicios resultante, define los servicios de los cuales estará compuesto el sistema, sus interacciones, y con que tecnologías serán implementados. Las interfaces que utiliza cada servicio para exponer su funcionalidad son gobernadas por contratos, que definen claramente el conjunto de mensajes soportados, su contenido y las políticas aplicables.

Visión interna de los servicios

Un servicio debe ser una aplicación completamente autónoma e independiente. A pesar de esto, no es una isla, porque expone una interfaz de llamado basada en mensajes, capaz de ser accedida a través de la red. Generalmente, los servicios incluyen tanto lógica de negocios como manejo de estado (datos), relevantes a la solución del problema para el cual fueron diseñados. La manipulación del estado es gobernada por las reglas de negocio(Parra 2007).

La comunicación hacia y desde el servicio, es realizada utilizando mensajes y no llamadas a métodos. Estos mensajes deben contener o referenciar toda la información necesaria para entenderlo. La idea es que haya el mínimo posible de llamadas entre el cliente y el servicio[2].
Un servicio es la evolución en complejidad de un componente distribuido, y se diferencian en:

- Mucho menos acoplados con sus aplicaciones cliente que los componentes.
- Menor granularidad que los componentes.
- No son diseñados e implementados necesariamente como parte de una aplicación end-to-end.
- Son controlados y administrados de manera independiente.
- Exponen su funcionalidad a través de protocolos abiertos e independientes de plataforma. Incluso arriesgando el rendimiento y consumo de recursos.
- Son transparentes de su localización en la red, de esta manera garantizan escalabilidad y tolerancia a fallos.
- Tienen sus propias políticas de escalabilidad, seguridad, tolerancia a fallos, manejo de excepciones y configuración.

La implementación ideal de un servicio exige resolver algunos inconvenientes técnicos inherentes a su modelo:

- Los tiempos de llamado no son despreciables, gracias a la comunicación de la red, tamaño de los mensajes. Esto necesariamente implica la utilización de mensajería confiable.

[2] Anexo 2: *Invocar el servicio de buscador de Google*

- La respuesta del servicio es afectada directamente por aspectos externos como problemas en la red, configuración. Estos deben ser tenidos en cuenta en el diseño, desarrollándose los mecanismos de contingencia que eviten la parálisis de las aplicaciones y servicios que dependen de él.
- Debe manejar comunicaciones no confiables, mensajes impredecibles, reintentos, mensajes fuera de secuencia.

La construcción de un servicio es una tarea mucho más complicada que la de un simple componente distribuido. El servicio debe publicar una interfaz (por ejemplo, utilizando WSDL) fácilmente localizable en la red. Esta interfaz debe servir como un contrato de servicio, donde se describen cada una de las funciones que provee, e incluso los niveles de prestación de servicio. Esta interfaz debe estar claramente documentada de manera que sea muy fácil implementar una conexión.

Un diseño exitoso de una arquitectura basada en servicios debe estar basado en una plataforma de mensajería confiable, que aísle de la implementación funcional muchos de los problemas anteriormente mencionados. Algunas de las responsabilidades de un mecanismo así, incluyen(Parra 2007):

- Entrega garantizada de mensajes.
- Enrutamiento de peticiones a un servicio disponible.
- Seguridad del contenido de los mensajes.
- QoS (Quality of Service) Calidad del Servicio.
- Escenarios fuera-de-línea.

Entre más de estas características tenga la tecnología elegida, menos problemas tendrá la solución final en operación. Una comunicación basada en mensajes por lo general implica que no existen sesiones. Por lo tanto, los clientes no guardan estado en el servicio (mayor escalabilidad), y la autenticación se debe dar a nivel de cada mensaje. Por último, los servicios deben ser diseñados para controlar internamente la transaccionalidad de sus propias operaciones. No es recomendable que una transacción traspase los límites de un servicio.

El problema de múltiples servicios

Cuando se usan múltiples servicios para implementar un sistema, es muy fácil que la comunicación entre estos se salga de control. Por ejemplo, se puede tener un servicio que llama a otros seis servicios, algunos de los cuales llama a otros servicios, y de esta manera, muy fácilmente el sistema se vuelve inmanejable. De esta manera, un sistema grande puede terminar con múltiples dependencias. Detectar un problema de rendimiento o funcionalidad se puede volver muy complicado(Parra 2007).

Una solución lógica a este problema es extraer los aspectos de procedimiento de varios servicios dentro de uno dedicado, llamado servicio de negocio. Un servicio de negocio controla las acciones paso a paso en la ejecución de algún trabajo, moviendo el sistema de un estado a otro. En cada paso, este llamara una operación de negocio provista por un servicio. Así, un servicio de negocio centraliza la definición del proceso, en vez de tener piezas del proceso entre todos los servicios. Esto disminuye las dependencias entre servicios y las aplicaciones clientes, ayudando a facilitar la administración del sistema.

Inconvenientes de los servicios web

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA.
- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI, CORBA, o DCOM. Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.
- Existe poca información de servicios web para algunos lenguajes de programación(Microsoft 2007).

Ventajas de los servicios web

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los servicios web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

- Al apoyarse en HTTP, los servicios web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.
- Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos.(Microsoft 2003)

Razones para crear un servicio web

La principal razón para usar servicios web es que se basan en HTTP sobre TCP en el puerto 80. Dado que las organizaciones protegen sus redes mediante firewalls que filtran y bloquean gran parte del tráfico de Internet, cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los servicios web se vehiculan por este puerto, por la simple razón de que no resultan bloqueados. Otra razón es que, antes de que existiera SOAP, no había buenas interfaces para acceder a las funcionalidades de otros ordenadores en red.

Una tercera razón por la que los servicios web son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el servicio web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más acusada(W3C 2006).

Organización estructural del sistema

Para desarrollar una solución SOA, primeramente se deberá plantear un ciclo de desarrollo. El “Ciclo de vida SOA” de IBM(IBM 2005), ha probado su efectividad a nivel mundial, tanto a grandes empresas como a pequeños proyectos. Sus fases están bien definidas y simplifican y organizan el trabajo. Básicamente las fases son: *Modelar,* donde se capturan los requisitos del negocio y se definen los servicios, *Ensamblar*, donde se crean los servicios ya definidos, Desplegar, para integrar dichos servicios, Administrar, para monitorear el rendimiento y funcionalidad de los servicios, y

organizando todo el proceso, una fase de *Gobernación* para coordinar políticas y estándares.

Hay que tener en cuenta que las arquitecturas orientadas a servicios se basan en un ecosistema de servicios que permite automatizar procesos de negocio que involucran más de un sistema de manera simple, ya que los mismos sistemas exponen su funcionalidad compartida a través de servicios. Por tanto el sistema propuesto estará estructurado en subsistemas o capas, que contendrá una capa de servicios compartidos, donde expondrá servicios y consumirá otros expuestos por otras aplicaciones

Capas

- Capa de Presentación: Da acceso al usuario a la aplicación, presenta los datos y permite la manipulación y entrada de datos.
- Capa de Servicios: Por ser independiente del cliente, puede ser movida de ubicación para atender requerimientos de tiempo y rendimiento. Provee servicios como seguridad, metadatos, definiciones de usuario, lógica de negocio y puede ser subdividida en varias capas entre el cliente y servidor.
- Capa de Persistencia: Usualmente es accedida por la capa de aplicación y en ocasiones, por la del usuario.

Como toda arquitectura que siga las especificaciones del patrón de capas(Lasso 2007), las capas superiores solicitarán servicios a las capas inmediatamente inferiores y estas a su vez a las siguientes, de esta manera cada nivel se responsabiliza con sus funciones. Las ventajas del esquema por capas son que provee reusabilidad, flexibilidad, capacidad de mantenimiento, capacidad de administración y escalabilidad. Además permite compartir y reutilizar componentes y servicios, así como dividir proyectos complejos y largos en varios más pequeños y más seguros de elaborar.

Presentación/Componentes de la Interfaz de Usuario (UI)

La capa de presentación es la responsable de interactuar con el usuario de la aplicación mediante una interfaz de usuario. Presenta los datos, Controla la navegación, recoge las entradas del usuario y las delega a la capa siguiente. Es decir, maneja el contexto del usuario y le permite interactuar con la capa de negocio donde está implementada toda la lógica de la aplicación(Lasso 2007). Si en particular

la lógica que se accede es compartida con otras aplicaciones lo hace a través de un servicio. Básicamente, los componentes y clases que se enmarcan dentro de este nivel, tienen como propósito servir de interfaz hombre-máquina con el usuario final del sistema.
La capa de presentación deberá ser implementada utilizando un Sistema de Gestión de Contenidos (en inglés Content Management System, abreviado CMS) ya que permite la creación y administración de contenidos principalmente en páginas Web.

CMS

Un CMS (Xavier Garcia 2004)consiste en una interfaz que controla una o varias bases de datos donde se aloja el contenido del sitio. El sistema permite manejar de manera independiente el contenido y el diseño. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio sin tener que darle formato al contenido de nuevo, además de permitir la fácil y controlada publicación en el sitio a varios editores. Un ejemplo clásico es el de editores que cargan el contenido al sistema y otro de nivel superior que permite que estos contenidos sean visibles a todo público. Las principales funcionalidades de un CMS son la creación y gestión de contenidos, su publicación y presentación.

Patrones

Para esta capa se recomienda organizar todas las clases utilizando el patrón Modelo Vista Controlador(Microsoft 2004). Este patrón permite separar responsabilidades entre las clases, lo que ayudará a la escalabilidad de la interfaz y la reutilización de componentes en futuros usos del sistema. Se busca cumplir la premisa "Un sistema escalable es aquel que permite agregar hardware y a la vez obtener mejoras en el funcionamiento."(Fowler 2002)

El MVC Se trata de un patrón de diseño en el que se manejan tres tipos de módulos:
- *Vista* - La capa de presentación, normalmente creada por los diseñadores gráficos, con la que interactúan los usuarios. Puede tener diferentes implementaciones dependiendo de la tecnología que se utilice para desarrollarla. Sin embargo los objetivos que se persiguen son los mismos. La usabilidad, diseño de interfaz amigable, validar la información que introduce el usuario y controlar el formato de presentación de la información.
- *Controlador* - La capa de lógica de negocios que recibe datos de la Vista, los procesa y le devuelve los resultados para que sean presentados de manera atractiva a los

usuarios. En este componente residirán varias funcionalidades. La más importante será el control del flujo de navegación entre las distintas vistas.

- *Modelo* – Es el objeto que representa los datos que manipula la aplicación. Aquí es donde suele residir la lógica de negocio.

Las ventajas que nos brinda este patrón son que hay una clara separación entre los componentes de un programa; lo cual permite implementarlos por separado. Hay un API muy bien definido, por lo que se podrá reemplazar el Modelo, la Vista o el Controlador sin dificultad. La conexión entre el Modelo y sus Vistas es dinámica, se produce en tiempo de ejecución. Básicamente, al incorporar el modelo MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Además, si uno de los componentes, posteriormente, se observa que funciona mal puede reemplazarse sin que los otros componentes se vean afectados.

Servicios/Componentes de la Lógica de la Aplicación

Esta capa o nivel dará servicio a la capa de presentación atendiendo a las necesidades que el usuario va requiriendo. La capa de servicios es la responsable de realizar las tareas para las cuales se diseña el sistema, es el contenedor lógico en que se ubican los servicios compartidos. Estos servicios exponen la funcionalidad compartida del sistema, tanto para el mismo sistema como otras aplicaciones.

Según la definición de SOA la lógica de la aplicación debe residir en servicios web accesibles a todas las aplicaciones consumidoras, la forma más común es mantener un registro de servicios mediante UDDI. Aunque la mayoría de las compañías lo utilizan con un bus empresarial de servicios (ESB) para la integración, siendo el principal componente de una arquitectura orientada a servicios, se propone prescindir de éste. En realidad muchas empresas desarrollan su propio ESB para adaptarlo a las prestaciones de sus modelos, pero este proceso implica mucho tiempo y esfuerzo, generalmente para cuando se ha implementado en concreto el ESB, ya las características del negocio pueden haber desviado sus requerimientos originales.
Por otra parte utilizar un ESB genérico es complicado por estar desarrollados en JAVA, ya que son componentes de software pesados, la curva de aprendizaje es lenta y se necesita escribir mucho código y desarrollar componentes para enlazarlos con otros que no son los desarrollados por el fabricante original. El UDDI ofrece la ventaja

de la rapidez de implementación y realiza las funciones de integración de servicios de manera eficaz.

UDDI

Son las siglas del catálogo de negocios de Internet denominado "*Universal Description, Discovery and Integration"*. UDDI es un registro público diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen. A través de UDDI, se puede publicar y descubrir información de una empresa y de sus servicios(Wolter 2001).

Lo más importante de UDDI es que contiene información sobre las interfaces técnicas de los servicios de una empresa. A través de un conjunto de llamadas a API XML basadas en SOAP, se puede interactuar con UDDI tanto en tiempo de diseño como de ejecución para descubrir datos técnicos de los servicios que permitan invocarlos y utilizarlos. De este modo, UDDI sirve como infraestructura para una colección de software basado en servicios web(Microsoft 2007).Los dos escenarios más comunes para UDDI dentro de una organización son la reutilización de código y la configuración dinámica de aplicaciones:

- Reutilización de código. En tiempo de diseño, los desarrolladores pueden buscar en UDDI los servicios web o recursos de programación para reutilizarlos en aplicaciones nuevas
- Configuración dinámica de aplicaciones. En tiempo de ejecución, una aplicación consulta el UDDI para descubrir la información actualizada de enlace al servicio que necesita, para conectarse después directamente a ese servicio.

La información de UDDI se aloja en nodos, que pueden ser privados o públicos. Por ejemplo, existen dos nodos públicos muy utilizados mundialmente que son los de Microsoft e IBM. Aunque una empresa puede implementar su propio UDDI para uso exclusivo de su intranet. Es importante observar que no existen requisitos de implementación para estos nodos, ya que puede utilizarse cualquier lenguaje o herramienta. Por ejemplo, el nodo de Microsoft está implementado en C# y utiliza Microsoft SQL Server, a su vez, IBM utiliza sus propias tecnologías(Microsoft 2002).

A la hora de utilizar estos recursos el cliente puede tener también sus propias tecnologías, por lo que UDDI representa la funcionabilidad en entornos heterogéneos que tanto se busca con el modelo de servicios web. Hay que tener en cuenta que UDDI se ha diseñado como registro, no como depósito. La diferencia esta en que un

registro redirige al usuario a recursos, mientras que un depósito sólo almacena información. Las consultas a UDDI conducen a una interfaz (un archivo .WSDL, .XSD, .DTD, entre otros.) o a una implementación (como un archivo .ASMX o .ASP) ubicadas en otro servidor

¿Qué información se aloja en UDDI?

Se definen cuatro tipos de datos básicos para la información del negocio y del servicio. Todos los datos en repositorio UDDI debe pertenecer a uno de estos cuatro tipos de datos. UDDI define cada tipo en una estructura de datos basada en XML y cada una contiene campos obligatorios y opcionales. Los cuatro tipos de datos base son:

- BusinessEntity
- TModel
- BusinessService
- BindingTemplate

La información de estos tipos de datos puede clasificarse en páginas blancas, amarillas y verdes.

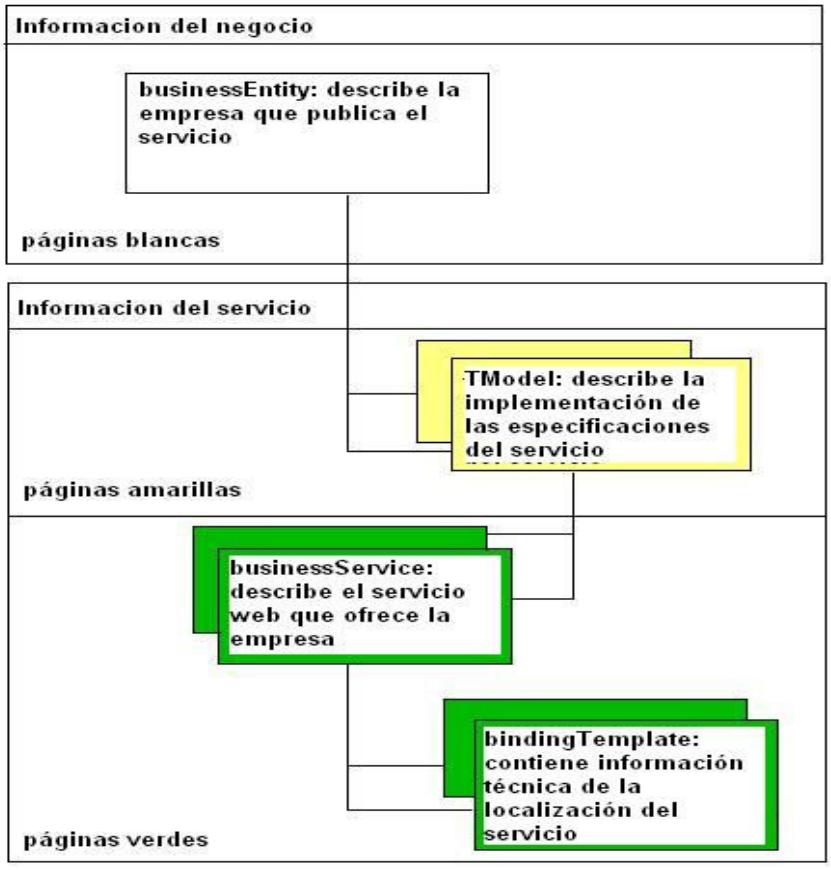

Fig. Información del UDDI

Información del Negocio

La información del negocio define el concepto de un elemento del tipo BusinessEntity. Durante la fase de descubrimiento, el elemento BusinessEntity funciona como un apuntador para recuperar la información pertinente acerca de la compañía y sus servicios.

Información del Servicio

La información del servicio la definen los elementos BusinessService y BindingTemplate. El elemento BusinessService almacena información técnica e información sobre la descripción de los servicios web ofrecidos por el negocio (páginas verdes).

UDDI permite que la información de un elemento BusinessService sea clasificado basado en producto, categoría o la combinación de ambos. Un elemento BusinessService contiene un elemento BindingTemplate que almacena la descripción técnica del servicio web.

Una estructura BusinessService contiene uno o más (1...n) BindingTemplates, las cuales son plantillas que enlazan el servicio web con su implementación actual. Se puede enlazar cada servicio con un número cualquiera de plantillas (BindingTemplates). Por lo tanto una plantilla debe contener dos piezas importantes de información. Primero una dirección o la localización del lugar donde se puede encontrar la implementación actual, y en segundo lugar, alguna descripción acerca de su arquitectura.

Cada estructura tModelInstanceInfo en una BindingTemplate se refiere a un tModel, esta es la segunda parte de la información que un BindingTemplate debe contener y es la parte más interesante de una estructura de datos UDDI. Los TModels, abreviatura de "Technology Model" (modelo de tecnología), representan huellas digitales técnicas, interfaces y tipos abstractos de metadatos. La creación del TModel requiere de algunos campos específicos como *name, desciription, overviewURL y categoryBag.* Los TModel son especificaciones técnicas que describen la naturaleza de un servicio, lo más adecuado es utilizar WSDL para describir estas especificaciones.

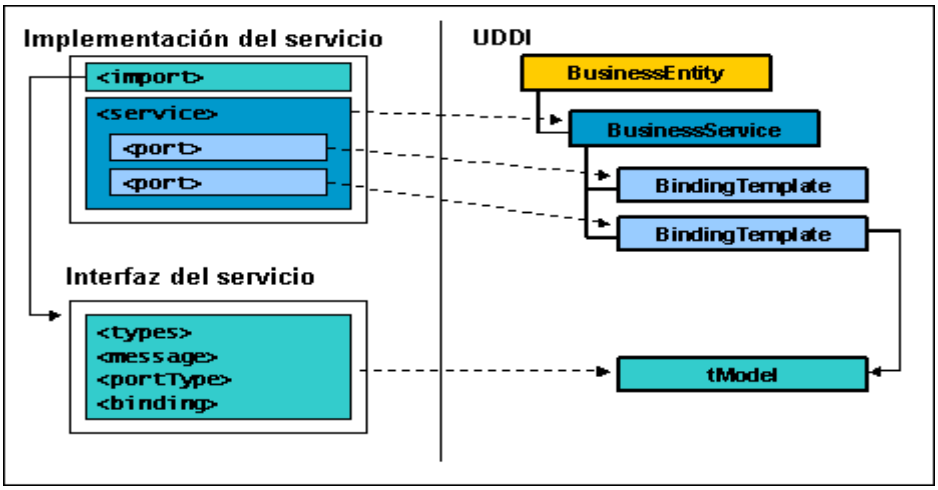

Fig. Tipos de datos UDDI

El elemento "BusinessService" puede contener otra información pertinente como opciones de ruteo para el balance de carga o una descripción de la información necesaria antes de invocar un servicio web. En pocas palabras, el elemento "BusinessService" contiene la información acerca de los servicios web ofrecidos, y el "BindingTemplate" contiene la información que se requiere para invocar el servicio.

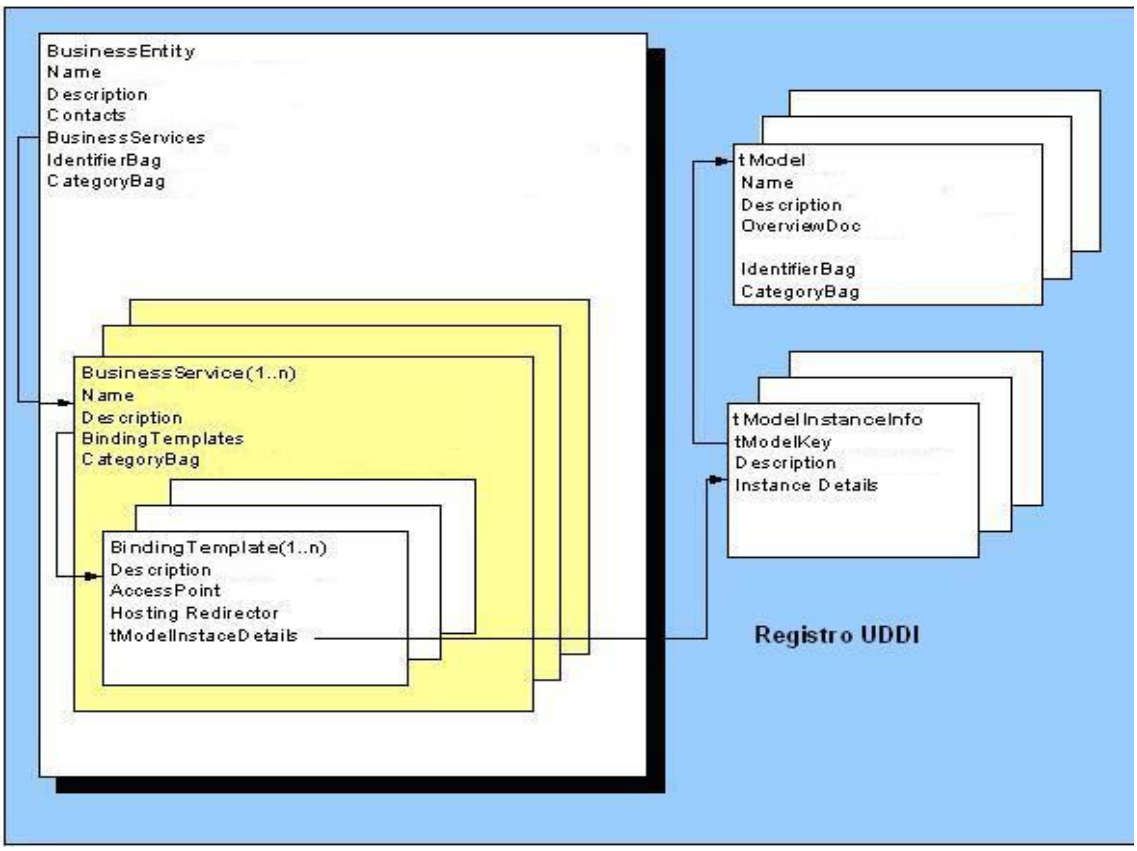

Fig. Un registro UDDI

¿Cómo funciona el UDDI?

La interfaz de programación de UDDI consta de dos partes. Una API de consulta para buscar/descubrir y un API para publicar/registrar. La especificación de UDDI no impone restricciones pero hace sugerencias acerca del diseño interno y la implementación de los registros. Esto significa que cualquier base de datos puede responder a las API de publicación y consulta en la forma que la especificación sugiere que lo haga UDDI.

Registrar en UDDI [3]

Cuando un consumidor de un servicio web desea interactuar con un Servicio Web, debe descargar el archivo WSDL del servicio. El archivo dice qué ofrece el servicio y cómo interactuar con el. En UDDI el vinculo para los WSDL de un Servicio Web deben estar ubicados en el atributo overviewURL del elemento overviewDoc de cada tModel que tiene un ServiceEntity (cada ServiceEntity es un servicio web).

La publicación en UDDI consiste en tres pasos(Januszewski 2003). El primer paso consiste en determinar información básica sobre cómo definir la empresa y los servicios en UDDI. El siguiente paso, una vez determinada esta información, consiste en llevar a cabo el registro, ya sea mediante programación o a través de una interfaz de usuario basada en el Web. Por último, se debe probar la entrada para asegurar que se registró correctamente y que aparece tal y como se esperaba en diferentes tipos de búsquedas y herramientas.

Paso 1: Definir la entrada de UDDI

- Determinar los TModels (archivos WSDL) que utilizan las implementaciones del servicio web.
- Determinar el nombre de la empresa y una breve descripción de la misma, así como los contactos principales para los servicios web que ofrece.
- Determinar las categorías e identificaciones adecuadas para la empresa.
- Determinar los servicios web que la empresa ofrece a través de UDDI.
- Determinar las categorías adecuadas para los servicios.

Paso 2: Registrar la entrada de UDDI

[3] Anexo 3: *Ejemplo de registro*

Paso 3: Buscar la entrada en UDDI

Descubrir servicios en UDDI [4]

Existen algunos métodos disponibles para buscar en el registro UDDI. Cualquier búsqueda en un registro UDDI(Bojacá 2003) es una combinación de estos métodos básicos de búsqueda.

- Encontrar el negocio de interés.
- Buscar todos los TModel de interés. El usuario debe proveer un criterio de búsqueda y el método retorna una lista de los TModels que cumplen este criterio.
- Encontrar los enlaces (bindings) de interés. Cuando se invoca este método el registro UDDI responde enviando una lista de todos los BindingTemplates que el servicio del negocio contiene. Los BindingTemplate ofrecen el punto de acceso del servicio web y referencia generalmente una interfaz WSDL.
- Obtener el detalle de un "BindingTemplate" en particular. Para obtener el detalle de un "BindingTemplate" en particular. Normalmente los usuarios invocan métodos *get* para recuperar detalles de los resultados abreviados de los métodos *find*.

En pocas palabras el proceso técnico para descubrir, encontrar e invocar un Servicio Web es:

- Usar el repositorio UDDI para localizar el elemento "BusinessEntity" del negocio apropiado
- Localizar el elemento "BusinessService" para identificar los servicios web ofrecidos por la empresa.
- Seleccionar el "BindingTemplate" para recuperar la dirección del Servicio Web y el elemento "TModel" para asegurar la compatibilidad técnica entre los sistemas.

UDDI en tiempo de ejecución

[4] Anexo 4: *Ejemplo de búsqueda*

Se descubre (busca) un “Web Service” en UDDI. Se utilizan el archivo WSDL de este servicio (representado por una entidad tModel de UDDI) y los detalles de la implementación, como el punto de acceso y cualquier otra información de configuración, que se incluyen en una estructura bindingTemplate.

Se prepara una aplicación cliente para el “Web Service”. En la aplicación cliente, se almacena en caché el valor único de bindingKey del servicio. Cuando la aplicación invoque el Web Service remoto, se utilizan los datos almacenados en caché que se obtuvieron del registro Web de UDDI. Si se produce un error en la invocación, se utilizan el valor de bindingKey y la llamada de la API get_bindingTemplate a un registro de UDDI para obtener una copia actualizada de la información de enlace.

Desventajas de UDDI

UDDI es por diseño y concepción un directorio que brinda algunas funcionalidades de búsqueda(Januszewski 2003), pero su meta principal es cumplir el rol de registro por lo tanto sus funcionalidades de búsqueda no son similares a las de los meta buscadores. UDDI a diferencia de sistemas como Google que se encarga por si mismo de recopilar su información y validarla confía en que las personas o empresas que cumplen con el rol de publicador, ingresan toda la información necesaria y consistente para que un servicio web sea accesible, no es responsabilidad de UDDI ni esta dentro de su alcance validar que esta información sea valida o siquiera accesible.

Ventajas de UDDI

- Posibilita compartir, buscar y reutilizar servicios web propios o de otros desarrolladores, así como otros recursos programables.
- Contribuye al aumento de la productividad en menor tiempo, ya que permite utilizar recursos ya elaborados.
- Disminuye el costo de recursos y desarrollo de aplicaciones.
- Contribuye al aumento de la confiabilidad de las aplicaciones, así como mejora el proceso de administración de estas.

En resumen UDDI es uno de los estándares básicos de los servicios web cuyo objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL, en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con los servicios web del catálogo de registros.

Servidor de aplicaciones

Un servidor de aplicaciones que provea de servicios a los clientes es decisivo en el modelo que se presenta. Debe incluir middleware o software de conectividad que le permita comunicarse con múltiples servicios y proporcione confiabilidad y seguridad(Linux 2007). Debe brindar una API o interfaz de programación de aplicaciones de manera que mantenga heterogeneidad de plataformas y soporte gran número de interfaces, así como variedad de estándares que le permita su funcionamiento en ambientes Web y conexión a gran variedad de fuentes de datos, sistemas y dispositivos. Entre los servicios habituales de un servidor de aplicaciones se incluyen los siguientes:

- Agrupación de recursos (por ejemplo, agrupación de conexiones de base de datos y agrupación de objetos)
- Administración de transacciones distribuida
- Comunicación asincrónica de programa, normalmente a través de colas de mensajes
- Un modelo de activación de objetos oportuno
- Interfaces de servicios web XML automáticas para tener acceso a objetos
- Servicios de detección de errores y estado de las aplicaciones
- Seguridad integrada

En resumen, un servidor de aplicaciones es una tecnología básica que proporciona la infraestructura y servicios clave a las aplicaciones alojadas en un sistema.

Patrones

Es posible que sobre la marcha sea necesario definir nuevos patrones de diseño a medida que se definan los servicios del negocio, no obstante es recomendable tener en cuenta los patrones Facade y Proxy a la hora de implementar los servicios web para así asegurar el menor acoplamiento posible.

Facade

El patrón de diseño Facade sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Hace que una biblioteca de software más fácil de usar y entender, ya que

implementa métodos convenientes para tareas comunes. Además reduce la dependencia de código externo en los trabajos internos de una librería.
"Cuando hay muchas dependencias entre clientes y las clases de implementación. Se utiliza facade para desacoplar los subsistemas de otros subsistemas y los propios clientes, promoviendo por tanto, independencia y portabilidad en los subsistemas."(Gamma Erich 1997)

El Facade se utilizará para la generación de componentes de negocio. Estos componentes implementan el patrón para la exposición de la lógica de negocios como servicios de alto desempeño de aplicaciones. La lógica de negocios es independiente de la tecnología que se utilice para exponer servicios. Además, puede agrupar servicios de un subsistema a través de un único componente, el cual realiza un uso óptimo del tráfico de red. Para arquitecturas orientadas a servicios, este componente se expone como “Web Service” para consumo directo desde otros subsistemas, procesos o aplicaciones cliente.

Proxy
Para adaptarse a las interfaces de aplicaciones externas se recomienda el uso del patrón Proxy ya que se utiliza como intermediario para acceder a un objeto, permitiendo controlar el acceso a él. El patrón Proxy utiliza una clase que implementa la misma interfaz que otro objeto, generalmente remoto. En la implementación de los métodos de la clase Proxy lo que realmente se hace es invocar al objeto real. De esta forma se encapsula el acceso al componente real, posiblemente externo y de más difícil acceso, dentro de una clase más cercana y accesible por los posibles clientes del Proxy.

Persistencia/Componentes de Almacenamiento de Datos

Esta capa es la encargada de gestionar el almacenamiento de los datos, generalmente en un sistema gestor de bases de datos relacionales, y en general de la comunicación del sistema con cualquier otro sistema que realice tareas auxiliares, como por ejemplo un middleware.

Los componentes de acceso a datos son una parte clave en el desarrollo del sistema. En una SOA la persistencia puede ser manejada directamente desde los servicios o utilizando un enfoque más correcto a través de software ORM (Object Relational Mapping) como motores o frameworks de persistencia(Fowler 2002).

Debido a la importancia que tiene la persistencia de la información en una aplicación se hace necesaria la unión de los modelos de bases de datos relacionales y de objetos. La utilización de una interfaz que una esos modelos es llamada mapeo de objetos relacional.
El ORM proporciona:

- Mapear clases a tablas: propiedad a columna, clase a tabla.
- Persistir objetos.
- Recuperar objetos persistidos.
- Recuperar una lista de objetos.

Además de bases de datos, los sistemas nuevos utilizan servicios expuestos por otras aplicaciones. Estos servicios son un origen de datos más, al cual se accede con un Proxy. Para la capa de negocio no existe diferencia entre leer un dato desde la base de datos o desde un servicio porque la capa de acceso a datos es la que se encarga de entregárselo.

Sistemas de gestión de base de datos

Los SGBD son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta(Garbage 2004). El propósito general de los SGBD es el de manejar de manera clara, sencilla y ordenada un conjunto de datos.
Los objetivos que debe cumplir un SGBD son los siguientes:

- Abstracción de la información
- Independencia
- Redundancia mínima
- Consistencia
- Seguridad
- Integridad
- Respaldo y recuperación
- Control de la concurrencia
- Tiempo de respuesta

Ventajas de un SGBD:

- Facilidad de manejo de grandes volúmenes de información.
- Gran velocidad en muy poco tiempo.
- Independencia del tratamiento de información.

- Seguridad de la información (acceso a usuarios autorizados), protección de información, de modificaciones, inclusiones, consulta.
- No hay duplicidad de información, comprobación de información en el momento de introducir la misma.
- Integridad referencial el terminar los registros.

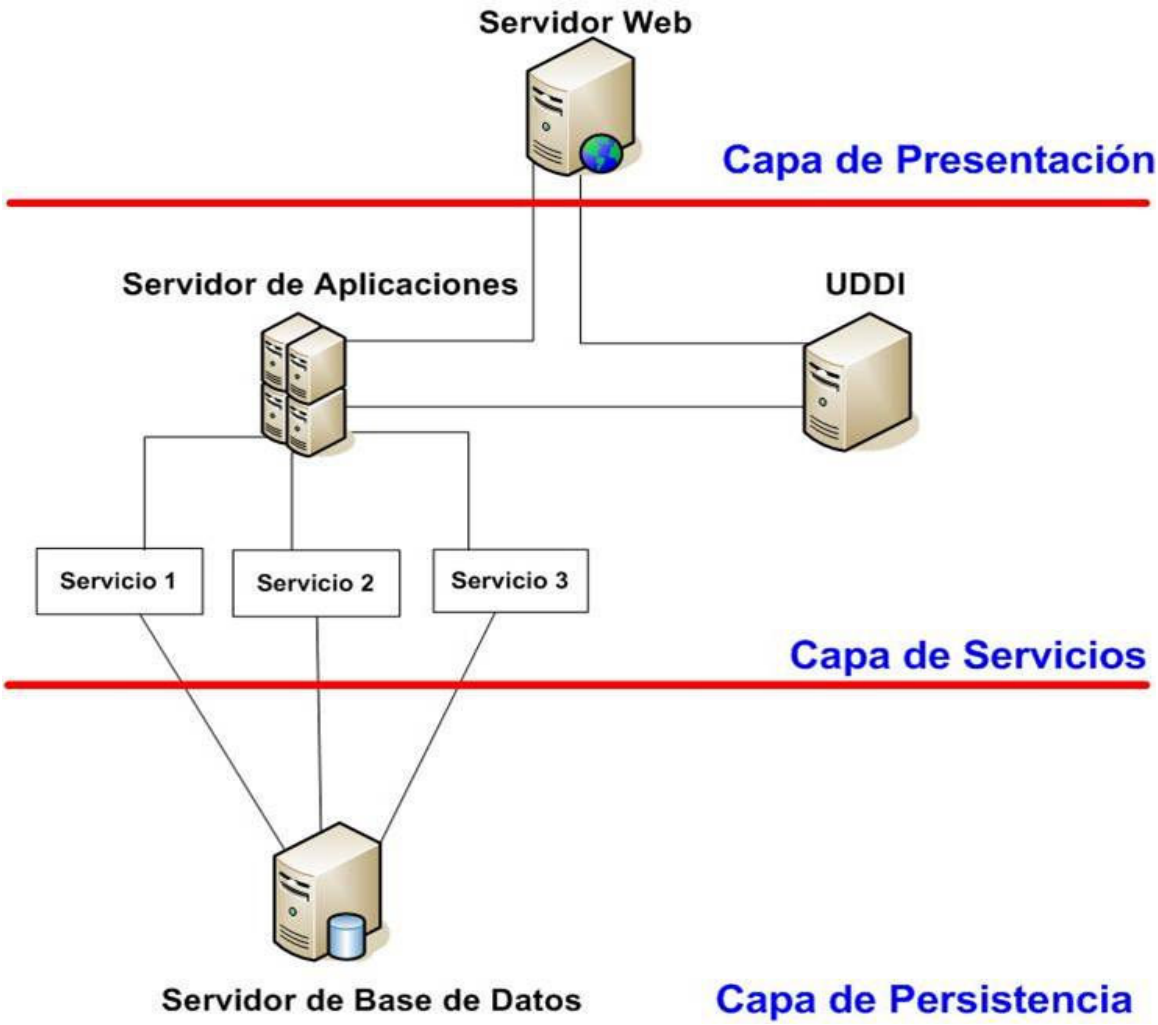

Fig. Modelo SOA propuesto

Propuesta de tecnologías a utilizar

Se realizó un estudio de las distintas tecnologías de software libre existentes que se acoplaban a los objetivos de la propuesta y se escogieron las que consideramos

óptimas para la implementación del modelo propuesto. Las tecnologías propuestas no son de uso obligatorio, existe un sin fin de tecnologías libres con las cuales se puede desarrollar una solución SOA, además según las características del sistema que se desee implementar, se pueden elegir otras a conveniencia de los desarrolladores.

Software libre y SOA

El estilo de arquitectura SOA replantea el uso de la tecnología tomando las mejores prácticas de las arquitecturas que la precedieron y de las actuales. No es un estilo de arquitectura que impulsa algún lenguaje en particular, sistema operativo o tecnología. A continuación se presentan algunas similitudes del software libre con algunos principios de SOA(Coletti 2006):

Reutilización

Este modo de crear tecnología (*open source*), impulsa que el código se pueda reutilizar, ya sea porque la licencia que gobierna el software exige que el código deba ser libre y pueda cambiarse (siempre y cuando la nueva aplicación mantenga las mismas libertades), o porque permite que el software una vez cambiado pueda ser cerrado y vendido como software propio. Teniendo el código a disposición y dependiendo el uso final que se le dará, solo resta conocer las libertades que ofrece la licencia para reutilizarlo o no. Es muy común entre los desarrolladores de *open source* reutilizar las cosas que funcionan bien, por lo que existe la garantía de que los componentes SOA que se utilicen ya hayan sido probado por una gran cantidad de usuarios, provocando una mayor solidez y estabilidad en el componente, así como una garantía libre de errores.

Interoperabilidad

La interoperabilidad es también una habilidad fuerte del software libre en general. Desde sus inicios, cuando se empezó con el proyecto GNU, el software debía interoperar con la mayor cantidad de sistemas operativos y otras aplicaciones. Esto fue una necesidad ya que no existían todos los componentes de un sistema operativo y aplicaciones libres preexistentes, con lo cual, este software siempre tuvo que interoperar con otras tecnologías. Esta característica es muy importante para el software libre ya que todo el software que se crea tiene como objetivo final que se utilice y para que se utilice, generalmente, tiene que interoperar con otro tipo de

software. Sea un sistema operativo no libre, una interfaz de usuario no libre o alguna base de datos o componente de un sistema no libre.

Uso de estándares

El software libre impulsa mucho el uso de estándares, no solo para poder comunicarse correctamente con software no libre, sino para poder comunicarse correctamente con cualquier tipo de software bien hecho. En un mercado tecnológico que crece constantemente y es cada día más diversificado, la implementación de estándares se convierte en algo cada vez más importante. Hasta las más grandes empresas de software entienden esta realidad y están adaptando los sistemas que venden para que implementen y usen estándares definidos por terceros.

Se propone el lenguaje *PHP* para desarrollar los servicios web, utilizando la herramienta *NuSOAP*. El CMS para la capa de presentación será *Mambo Open Source.* El servidor de aplicaciones *Apache* para los servicios, el *jUDDI* para integrar y el *PostgreSQL* como sistema gestor de bases de datos con framework *EZPDO*.

Mambo Open Source

Es un portal para manejo de contenidos y para la creación de sitios web. Es una aplicación escrita en lenguaje PHP y emplea un sistema de base de datos MySQL. Basa todo su aspecto en *templates* o *themes*. Permite la creación y mantenimiento de sitios web y portales de manera fácil y dinámica, permitiendo al dueño o administrador de una página web la simplicidad para actualizarla y hacerla accesible a todo tipo de usuarios a través de una variedad de instrumentos. La simplicidad de Mambo radica en que no son necesarios conocimientos técnicos ni especializados para crear, mantener, actualizar o personalizar los contenidos de un sitio web.

Características principales:

- Administración sencilla con atractiva interfaz gráfica
- Módulo de seguridad multinivel para usuarios/administradores
- Noticias, productos o secciones totalmente editables y configurables
- Sección de temas que pueden ser enviados por los usuarios registrados
- Foros dinámicos y encuestas con vista de resultados
- Multiplataforma, soporta Linux, FreeBSD, MacOSX, Solaris, AIX, SCO, WinNT, Win2K

- Gestión y Administración de usuarios registrados
- Generador automático de noticias en titulares
- Búsqueda Automática de Directorios
- Sindicación de Noticias.
- Posibilidad de impresión, convertidor a PDF o envío por email de cualquier noticia o artículo publicado.
- Editor de texto, similar al Word Pad
- Editor de Usuarios
- Módulos configurables. Descarga de nuevos módulos
- Administrador de Plantillas. Descarga de nuevas plantillas
- Zonas personalizables por el usuario
- Servicio de cuentas online
- Motor de búsqueda integrado
- Sistema de generación de noticias para ofrecerlas vía XML (formato RSS/RDF) automáticamente en otros sitios web
- Soporte para más de 20 lenguajes

PHP

Es un lenguaje de programación usado frecuentemente para la creación de contenido para sitios web con los cuales se puede programar las páginas html y los códigos de fuente. PHP es un acrónimo recursivo que significa "PHP Hypertext Pre-processor" (inicialmente PHP Tools, o, Personal Home Page Tools), y se trata de un lenguaje interpretado usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios web.

El fácil uso y la similitud con los lenguajes más comunes de programación estructurada, como C y Perl, permiten a la mayoría de los programadores experimentados crear aplicaciones complejas con una curva de aprendizaje muy suave. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones y prácticas.

Su interpretación y ejecución se da en el servidor web, en el cual se encuentra almacenado el script, y el cliente sólo recibe el resultado de la ejecución. Cuando el cliente hace una petición al servidor para que le envíe una página web, generada por un script PHP, el servidor ejecuta el intérprete de PHP, el cual procesa el script solicitado que generará el contenido de manera dinámica, pudiendo modificar el contenido a enviar, y regresa el resultado al servidor, el cual se encarga de

regresárselo al cliente. Además es posible utilizar PHP para generar archivos PDF, Flash, así como imágenes en diferentes formatos, entre otras cosas.

Permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, Postgres, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird y SQLite; lo cual permite la creación de Aplicaciones web muy robustas. PHP también tiene la capacidad de ser ejecutado en la mayoría de los sistemas operativos tales como UNIX (y de ese tipo, como Linux), Windows y Mac OS X, y puede interactuar con los servidores de web más populares ya que existe en versión CGI, módulo para Apache, e ISAPI.

Los principales usos del PHP son los siguientes:

- Programación de páginas web dinámicas, habitualmente en combinación con el motor de base datos MySQL, aunque cuenta con soporte nativo para otros motores, incluyendo el estándar ODBC, lo que amplía en gran medida sus posibilidades de conexión.
- Programación en consola, al estilo de Perl o Shell scripting.
- Creación de aplicaciones gráficas independientes del navegador, por medio de la combinación de PHP y GTK (GIMP Tool Kit), lo que permite desarrollar aplicaciones de escritorio en los sistemas operativos en los que está soportado.

Ventajas

- Es un lenguaje multiplataforma.
- Capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL
- Leer y manipular datos desde diversas fuentes, incluyendo datos que pueden ingresar los usuarios desde formularios HTML.
- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados ext's o extensiones).
- Posee una amplia documentación en su página oficial ([1]), entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.

- Permite las técnicas de Programación Orientada a Objetos.
- Permite crear los formularios para la web.
- Biblioteca nativa de funciones sumamente amplia e incluida
- No requiere definición de tipos de variables ni manejo detallado del bajo nivel

Servidor Apache

Hoy en día es el servidor web más utilizado del mundo, encontrándose muy por encima de sus competidores, tanto gratuitos como comerciales. Es un software de código abierto que funciona sobre cualquier plataforma(linalco 2005).

Tiene capacidad para servir páginas tanto de contenido estático, para lo que serviría sencillamente un ordenador 486, como de contenido dinámico a través de otras herramientas soportadas que facilitan la actualización de los contenidos mediante bases de datos, ficheros u otras fuentes de información. Apache es uno de los servidores Web más potentes del mercado, ofreciendo una perfecta combinación entre estabilidad y sencillez. Las principales características de Apache son:

- Funcionalidad en múltiples plataformas.
- Elaborado indicie de directorios.
- Soporte del último protocolo http 1.1.
- Sencilla administración basada en la configuración de un único archivo.
- Soporte para CGI (Common Gatewav Interface) y FastCGI

Apache está diseñado para ser un servidor web potente y flexible que pueda funcionar en la más amplia variedad de plataformas y entornos. Las diferentes plataformas y entornos, hacen que a menudo sean necesarias diferentes características o funcionalidades. Apache se ha adaptado siempre a una gran variedad de entornos a través de su diseño modular. Este diseño permite a los administradores de sitios web elegir que características van a ser incluidas en el servidor seleccionando que módulos se van a cargar, ya sea al compilar o al ejecutar el servidor.

jUDDI

Es una implementación en Java de un UDDI para servicios web. Entre sus ventajas destaca su capacidad de ejecutarse en diferentes sistemas operativos, o sea es multiplataforma. Es open source, por lo que se le pueden realizar cambios para

adaptarlo a las necesidades propias del usuario. Se puede usar con gestores de bases de datos como PostreSQL, MySQL, DB2, Sybase, entre otros.

NuSOAP

Es un kit de herramientas (ToolKit) para desarrollar “Web Services” bajo el lenguaje PHP. Está compuesto por una serie de clases que facilitan el desarrollo de “Web Services”. Provee soporte para el desarrollo de clientes consumidores de servicios y de servidores proveedores de estos. Entre sus principales ventajas está que no necesita módulos adicionales y es de fácil instalación y uso, y además va en desarrollo progresivo.

PostgreSQL

Es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS por sus siglas en inglés). PostgreSQL está ampliamente considerado como el sistema de bases de datos de código abierto más avanzado del mundo. Posee la mayoría de las características presentes en grandes sistemas gestores de bases de datos comerciales, tales como transacciones, subconsultas, gatillos (triggers), vistas, integridad referencial con llaves externas, y bloqueo sofisticado. También tenemos algunas características que no tienen las otras, como tipos definidos por el usuario, herencia, reglas, y control de concurrencia multi-versión para reducir el bloqueo de controversias.

Entre sus características más importantes se pueden resaltar:

- Alta concurrencia. Mediante un sistema denominado MVCC (Acceso concurrente multiversión) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos.
- Claves ajenas también denominadas Llaves ajenas o Llaves Foráneas (foreign keys).
- Disparadores (triggers).
- Vistas.
- Integridad transaccional.
- Herencia de tablas.
- Tipos de datos y operaciones geométricas.

EZPDO

"Easy PHP Data Objects" (EZPDO) es una solución de persistencia para PHP. Es ligera y fácil de usar y sus desarrolladores se basan en la simplicidad como llave primaria. Entre sus ventajas están:

- No requiere apenas conocimientos de SQL
- Trabaja con bases de datos y codigo existentes
- Requiere especificaciones ORM mínimas
- No necesita de compilador explicito de líneas de comando
- Posee poco tiempo de respuesta para garantizar su funcionamiento
- Soporta Object Query (EZOQL)
- Auto genera tablas de bases de datos

Otras tecnologías open source disponibles

Para la capa de presentación existen muchas opciones a la hora de escoger un CMS que se adapte a las necesidades de los desarrolladores y el sistema a desarrollar, lo más importante es que permita la creación de componentes dinámicos. Entre los más destacados se encuentran Zope/Plone, Drupal y Typo3, entre otros.

En la capa de servicios para crear servicios web se pueden escoger otros lenguajes de programación, siempre y cuando soporten "web services", por ejemplo Java, además se ha probado internacionalmente la utilidad de algunas herramientas como Apache Axis, Struts, Ajax, y otros servidores de aplicaciones pueden ser GlassFish y el JBoss. Para integrar los servicios tambien se puede ver el Mule, Service Mix y el Celtic.

Para la capa de persistencia como sistema gestor de bases de datos existen diversas herramientas como MySQL, en conjunto con frameworks ORM como Hibernate, Dejavu, Cayenne, Apache OJB, entre otros.

Conclusiones

Este modelo está propuesto para el desarrollo de aplicaciones en la fábrica de portales de la Facultad 10 de la Universidad de las Ciencias Informáticas, en contraste con los modelos que se venden y utilizan a nivel internacional de probada eficiencia pero desarrollados por grandes compañías y empresas que se dedican a brindar soluciones, soporte y tecnologías a altos precios para estructuras orientadas a servicios. Se ha elaborado esta propuesta fundamentándose en la necesidad de la utilización de software libre y la poca o mediana envergadura de los sistemas que aquí se desarrollan, así como la privacidad de los recursos internos del centro. Es un modelo con una estructura sencilla, lo que facilita su implementación de manera eficaz en un tiempo relativamente corto, y es esta su principal característica y ventaja con respecto a grandes modelos pensados para negocios con numerosos procesos y recursos.

Este modelo no esta pensado para aplicaciones en tiempo real, procesos sincrónicos que tengan un tiempo de respuesta cercano a la inmediatez o que en su modelo de despliegue no incluya la infraestructura necesaria de servidores o red.

Recomendaciones

Se recomienda profundizar en el estudio de la propuesta para implantarla en los sistemas que se desarrollan actualmente en la Universidad de las Ciencias Informáticas con el objetivo de fomentar la creación de servicios web y mejorar así las prestaciones que aquí se ofrecen. Hay que tener en cuenta para una migración a este modelo, realizar un análisis profundo de las particularidades de cada proyecto como los requerimientos que poseen y la arquitectura que tienen implementada. Para el óptimo desarrollo de la orientación a servicios en la UCI, se recomienda la creación de un catálogo de registros de servicios o UDDI, en el cual se puedan registrar todos los servicios que se ofrecen al nivel de la infraestructura de producción. Una vez que se comiencen a compartir los servicios será posible buscar el servicio necesario y asi reutilizarlo, no importa cómo esté implementado ni qué plataforma utilice su proveedor. Hay que tener en cuenta que la reutilización de servicios puede mejorar de forma práctica la producción de la universidad asi como disminuir el costo de recursos y el tiempo de desarrollo de nuevos sistemas y aplicaciones.

Bibliografía

(AG 2007) Sitio oficial de Software AG. (2007). disponible en http://www.softwareag.com/es/default.asp.

(Barco 2006) Barco, A. (2006). "Principios de la orientación a servicios." disponible en http://arquitecturaorientadaaservicios.blogspot.com/2006_06_01_archive.html.

""WSDL: El contrato de un Servicio"." disponible en http://arquitecturaorientadaaservicios.blogspot.com/.

(Bojacá 2003) Bojacá, J. T. (2003). "Motor de Búsqueda sobre UDDI." disponible en www.willydev.net/Descargas/Articulos/jaimetarquino/uddi.pdf

(Booch Grady 1998) Booch Grady, J. I., Rumbaugh James (1998). The Unified Software Development Process, Addison-Wesley.

(Chapell 2007) Chapell, D. (2007). "SOA es la respuesta para evolucionar las TI de forma rápida y sencilla." disponible en http://www.microsoft.com/spain/enterprise/perspectivas/numero21/tendencias.mspx.

(Coletti 2006) Coletti, D. (2006). "SOA y el Software Libre." disponible en http://www.danielcoletti.com.ar/index.php/2006/11/17/soa-y-el-software-libre/.

(Erik Christensen 2000) Erik Christensen, F. C., Greg Meredith, Sanjiva Weerawarana. (2000). "Lenguaje de descripción de servicios Web (WSDL) 1.0." disponible en http://www.microsoft.com/spanish/msdn/articulos/archivo/090201/voices/wsdl.asp.

(Fernández 2007) Fernández, R. (2007). "IBM apuesta fuerte a SOA en la Región." disponible en http://www.cwv.com.ve/index.php?option=com_content&task=view&id=222&Itemid=1.

(Fowler 2002) Fowler, M. (2002). Patterns of Enterprise Application Architecture, Addison-Wesley.

(IBM 2005) IBM (2005). IBM SOA Foundation: providing what you need to get started with SOA, IBM. disponible en http://t1d.www03.cacheibm.com/industries/media/doc/content/bin/SOA_Foundation_G224-7540-00.pdf

(IBM 2007) Sitio oficial de IBM (español). (2007). disponible en http://www.ibm.com/es/.

Sitio official de IBM (english). (2007). disponible en http://www.ibm.com/us/.

"Conozca SOA." disponible en http://www-306.ibm.com/e-business/la/ar/soa_site/soa_2.shtml.

"Infraestructura y software." disponible en http://www-306.ibm.com/e-business/la/ar/soa_site/infrastructure.shtml.

"Service Oriented Architecture." disponible en http://www-306.ibm.com/software/solutions/soa/.

"Service Oriented Architecture (SOA) Entry Points." disponible en http://www-306.ibm.com/software/solutions/soa/entrypoints/index.html?

"SOA – it's about business." disponible en http://www-935.ibm.com/services/us/index.wss/whitepaper/gbs/a1026003?cntxt=a1002583.

"SOA and Web services." disponible en http://www-128.ibm.com/developerworks/webservices.

"SOA Consulting Services." disponible en http://www-306.ibm.com/software/solutions/soa/services.html.

(James McGovern 2003) James McGovern, S. W. A., Michael E. Stevens, James Linn, Vikas Sharan, Elias K. Jo (2003). A Practical Guide to Enterprise Architecture, Prentice Hall PTR.

(Januszewski 2003) Januszewski, K. (2003). "Descripción y descubrimiento de servicios Web con UDDI, primera parte." disponible en http://www.microsoft.com/spanish/msdn/articulos/archivo/281201/voiceszservice10032001.asp.

(Lasso 2007) Lasso, A. (2007). "Arquitectura de Software." disponible en http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art110.asp.

(Len Bass 1997) Len Bass, P. C., Rick Kazman (1997). Software Architecture in Practice, Addison-Wesley.

(linalco 2005) linalco. (2005). "Servidor Web Apache." disponible en http://www.linalco.com/apache.html.

(Linux 2007) Linux, p. t. (2007). "Servidor Web ", disponible en http://www.linuxparatodos.net/portal/staticpages/index.php?page=servidor-web.

(Luke 2003) Luke, H. (2003). Beyond Software Architecture: Creating and Sustaining Winning Solutions, Addison Wesley.

(Microsoft 2002) Microsoft. (2002). "Preguntas más frecuentes sobre UDDI." disponible en http://www.microsoft.com/spain/windowsserver2003/technologies/idm/uddi/uddifaq.mspx.

(Microsoft 2003) Microsoft. (2003). "Las 10 ventajas principales de Windows Server 2003." disponible en http://www.microsoft.com/latam/windowsserver2003/evaluation/whyupgrade/top10best.mspx.

(Microsoft 2005) Microsoft. (2004). "Estilos y Patrones en la Estrategia de Arquitectura de Microsoft." disponible en http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp.

(Microsoft 2006) Microsoft. (2006). "Real-World SOA through the Microsoft Platform." disponible en http://www.microsoft.com/biztalk/solutions/soa/whitepaper.mspx.

(Microsoft 2007) Sitio oficial de Microsoft Corporation (español). (2007). disponible en http://www.microsoft.com/spanish/.

Sitio oficial de Microsoft Corporation (english). (2007). disponible en http://www.microsoft.com/en/us/default.aspx.

"Microsoft BizTalk Server." disponible en http://www.microsoft.com/spain/biztalk/default.mspx.

"MSDN Solutions Architecture Center." disponible en http://msdn2.microsoft.com/en-gb/architecture/default.aspx.

"Service Oriented Architecture." disponible en http://msdn2.microsoft.com/en-gb/architecture/aa948857.aspx.

"UDDI Services." disponible en http://www.microsoft.com/windowsserver2003/technologies/idm/uddi/default.mspx.

"Ventajas de los servicios Web XML creados mediante ASP.NET." disponible en http://msdn2.microsoft.com/es-es/library/0859ebft(vs.80).aspx.

"Web Services and Other Distributed Technologies." disponible en http://msdn2.microsoft.com/en-gb/webservices/default.aspx.

"Web Services Basics." disponible en http://msdn2.microsoft.com/en-us/webservices/aa740691.aspx.

(Nilo Mitra 2006) Nilo Mitra, Y. L. (2006). "SOAP Version 1.2 Part 0: Primer (Second Edition)." disponible en http://www.w3.org/TR/soap12-part0/.

(Parra 2007) Parra, J. D. (2007). "Hacia una Arquitectura Empresarial basada en Servicios." disponible en http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art143.asp.

(Paul Clements 2002) Paul Clements, F. B., Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford (2002). Documenting Software Architectures: Views and Beyond, Addison Wesley.

(Peña 2005) Peña, J. R. (2005). "Contamos con la gama de prestaciones de administración e integración de procesos más amplia del mercado actual." disponible en http://www.cwv.com.ve/index.php?option=com_content&task=view&id=78&Itemid=39.

(Shodjai 2006) Shodjai, P. (2006). "Web Services and the Microsoft Platform." disponible en http://msdn2.microsoft.com/En-US/library/aa480728.aspx.

(W3C 2006) W3C. (2006). "Guía Breve de Servicios Web." disponible en http://www.w3c.es/Divulgacion/Guiasbreves/ServiciosWeb.

"Web Services Activity." disponible en http://www.w3.org/2002/ws/.

(Wolter 2001) Wolter, R. (2001). "Fundamentos de los servicios Web XML." disponible en http://www.microsoft.com/spanish/msdn/articulos/archivo/151102/voices/fundamentos_xml.asp.

(WS-I 2006) WS-I. (2006). disponible en http://www.ws-i.org/.

(XML 2007) XML. (2007). disponible en http://www.xml.org/.

Bastarrica María Cecilia , G. D., Wilckens Cristian (2005). Input/Output autómatas como lenguaje de definición de arquitecturas. Facultad de Ingeniería, Universidad de Tarapacá. disponible en http://www.scielo.cl/pdf/rfacing/v13n1/art09.pdf

Beack, T. (2006). Primeros Pasos: Su Guía al Éxito con SOA, Software AG. disponible enhttp://www.softwareaglatam.com/mexico/SOA/imgsnews/Guia_hacia_el_exito_con_SOA.pdf

Bednarz, A. (2007). "10 best practices for your enterprise SOA." disponible en http://www.networkworld.com/supp/2007/ndc1/021907-ndc-best-of-enterprise-soa.html.

Cárdenas, D. S. Á. (2005). "Diseño de la arquitectura y los servicios Web." disponible en http://revistas.mes.edu.cu/01-Libros-por-ISBN/959-16-0400/0318_Uciencias/.

Casanovas, J. (2004). "Usabilidad y arquitectura del software." disponible en http://www.alzado.org/articulo.php?id_art=355.

Casanovas, J. (2007). "¿Qué es la Arquitectura del Software?" disponible en http://www.wikilearning.com/que_es_la_arquitectura_del_software-wkccp-4151-3.htm.

Daniel Hiebert, R. A. K., Elena Lowery, Aleksandr Nartovich, Nitin Raut, Michael J. Sandberg (2007). Building SOA-based Solutions for IBM System i Platform, IBM. disponible en http://www.redbooks.ibm.com/redpieces/pdfs/sg247284.pdf

David Sprott, L. W. (2004). "Understanding Service-Oriented Architecture." The Architecture Journal, disponible en http://msdn2.microsoft.com/en-us/library/aa480021.aspx.

Fernández, B. (2007). "La Importancia de la Gobernabilidad en SOA." disponible en http://espaciosoa.wordpress.com/2007/04/02/la-importancia-de-la-gobernabilidad-en-soa/.

Gamma Erich, H. R., Johnson Ralph, Vlissides John (1997). Design Patterns, Addison-Wesley.

ITprofessionals. (2006). "SOA: una arquitectura para la empresa del S. XXI." disponible en http://www.itprofessionals.es/detalle_noticia.asp?Id=145.

Martin Keen, A. A., Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, Paul Verschueren (2004). Patterns: Implementing an SOA Using an Enterprise Service Bus, IBM. disponible en http://www.redbooks.ibm.com/redpieces/pdfs/sg246346.pdf

Network, W. (2006). "SOA (service-oriented architecture)." disponible en http://www.networkworld.com/details/6187.html.

OASIS. (2006). disponible en http://www.oasis-open.org/home/index.php.

Organero, M. M. (2006). "Sistemas de información. Servicios web." disponible en www.it.uc3m.es/mcfp/docencia/si/material/12_WS_II_mcfp.pdf.

Pablo, J. (2007). "ABC de la arquitectura SOA + 3 capas." disponible en http://liarjo.spaces.live.com/blog/cns!4131EA552C5BB029!1206.entry.

Patterns, S. E. (2006). "Services, Orchestration, and Beyond." disponible en http://orchestrationpatterns.com/.

Sánchez, L. M. (2007). Utilización de Patrones para la Construcción de Arquitecturas Orientadas a Servicios. Facultad de Informática. Madrid, Universidad Pontificia de Salamanca. disponible en http://www.luismor.com/UtilizaciondePatronesParaLaConstrucciondeArquitecturasOrientadasAServicios.pdf

SOA. (2007). disponible en http://www.soa.com/.

UDDI. (2006). disponible en http://www.uddi.org/.

WebServices. (2005). disponible en http://www.webservices.org/.

ANEXOS

Anexo 1: HelloService.wsdl

Crea el documento WSDL de un servicio “helloservice” con la funcion sayHello.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
  targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
  </message>
  <message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>

  <portType name="Hello_PortType">
    <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
    </operation>
  </portType>

  <binding name="Hello_Binding" type="tns:Hello_PortType">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

```
            namespace="urn:examples:helloservice"
            use="encoded"/>
        </input>
        <output>
          <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:examples:helloservice"
            use="encoded"/>
        </output>
      </operation>
    </binding>

    <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
        <soap:address
          location="http://localhost:8080/soap/servlet/rpcrouter"/>
      </port>
    </service>
</definitions>
```

Anexo 2: Invocar el buscador Google como servicio

Para utilizar el servicio de buscador Google con NuSOAP

```
#!/usr/local/bin/php -q
<?php
require_once('nusoap-0.6.1/nusoap.php');

////////// Licencia de Google
$key = 'iwnUXUtHj3bteg5FWfBJDwui3SPeB+iy';
//////////query de ejemplo
$Query             = "web service+php+NuSOAP";
$startPage         = 0;
////////// Se Definen los parámetros de entrada
$parameters = array(

          'key'         => $key,
          'q'           => $Query,
          'start'       => $startPage,
          'maxResults'  => 10,
          'filter'      => false,
          'restrict'    => '',
          'safeSearch'  => false,
          'lr'          => '',
          'ie'          => '',
          'oe'          => ''
);

////////// Se indica la URL del servicio SOAP mediante WSDL
$soapclient = new soapclient('http://api.google.com/search/beta2');
////////// Se invoca al Servicio Web SOAP
$result = $soapclient->call('doGoogleSearch',$parameters, 'urn:GoogleSearch');
print_r ($result);
?>
```

Anexo 3: Ejemplo de registro

A partir del documento WSDL del Anexo 1, con este bindingTemplate se publica el servicio "*helloservice"* en un UDDI.

```
<businessService businessKey="..." serviceKey="...">
  <name>helloService</name>
  <description> (...) </description>
  <bindingTemplates>
    <bindingTemplate>
      (...)
      <accessPoint urlType="http">
        http://example.com/helloservice
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="...">

        </tModelInstanceInfo>
      <tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
</businessService>
```

Anexo 4: Ejemplo de búsqueda

Mensaje enviado a un registro UDDI para buscar servicios llamados "*helloservice*".

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <find_service businessKey="*" generic="1.0" xmlns="urn:uddi-org:api">
      <name>helloservice</name>
    </find_service>
  </Body>
</Envelope>
```

GLOSARIO

AJAX: AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript y XML asíncronos), es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el cliente, es decir, en el navegador de los usuarios, y mantiene comunicación asíncrona con el servidor en segundo plano.

API: Una API (del inglés Application Programming Interface - Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

Arquitectura en PIPELINE: La arquitectura en pipeline (basada en filtros) consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior.

Arquitectura en PIZARRA: La arquitectura software en pizarra es un modelo arquitectónico de software habitualmente utilizado en sistemas expertos, sistemas multiagente y, en general, sistemas basados en el conocimiento.

BluePrint: Una Blueprint es una reproducción documentada sobre dibujos y diagramas técnicos de arquitectura e ingeniería de diseño.

Bottom-up y Top-down: Top-down y Bottom-up son estrategias de procesamiento de información más que todo en las ciencias de la información, involucrando software. El modelo "Top-down" se diseña con frecuencia con la ayuda de "cajas negras" que hacen más fácil cumplir requerimientos aunque estas cajas negras no expliquen en detalle los componentes individuales. En contraste, en el diseño “Bottom-up” las partes individuales se diseñan con detalle y luego se enlazan para formar componentes más grandes, que a su vez se enlazan hasta que se forma el sistema completo.

DCOM: Distributed Component Object Model (DCOM), en español Modelo de Objetos de Componentes Distribuidos, es una tecnología propietaria de Microsoft para desarrollar componentes software distribuidos sobre varios ordenadores y que se comunican entre sí.

CORBA: En computación, CORBA (Common Object Request Broker Architecture — arquitectura común de intermediarios en peticiones a objetos), es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

Drivers: Un dispositivo o software "driver" es un programa de computación que permite a otros programas interactuar con un dispositivo "hardware" de una computadora.

Entorno de escritorio: Un entorno de escritorio (en inglés, Desktop Environment) es un conjunto de software para ofrecer al usuario de un ordenador un ambiente amigable y cómodo.

Firewall: Un cortafuegos (o firewall en inglés), es un elemento de hardware o software utilizado en una red de computadoras para controlar las comunicaciones, permitiéndolas o prohibiéndolas según las políticas de red que haya definido la organización responsable de la red.

Fujitsu: Es una compañía japonesa especializada en el área de los semiconductores, computadoras (supercomputadoras, computadoras personales, servidores), telecomunicaciones, y servicios.

Google: Google Inc., empresa propietaria de la marca Google, es una compañía cuyo principal producto es el motor de búsqueda del mismo nombre.

Granularidad: Es una medida del tamaño de un componente o descripción de un componente que conforman un sistema.

Herramientas CASE: Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

HTTP: El protocolo de transferencia de hipertexto (HTTP, HyperText Transfer Protocol) es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante

el cual se envían las peticiones de acceso a una página y la respuesta con el contenido.

IIOP: Internet Inter-ORB Protocol, es un protocolo desarrollado por el OMG (Object Management Group) para hacer posible la comunicación en Internet entre los programas distribuidos escritos en diferentes lenguajes.

Infraestructura Tecnológica: En una organización la IT no es más que el conocimiento de los trabajadores en la disponibilidad y en el funcionamiento de las herramientas requeridas para operar y compartir este conocimiento.

Interfaz: En software, una interfaz de usuario es la parte del programa informático que permite el flujo de información entre varias aplicaciones o entre el propio programa y el usuario.

International Business Machines o IBM: Conocida coloquialmente como el Gigante Azul, es una empresa que fabrica y comercializa hardware, software y servicios relacionados con la informática.

Internet: Es un método de interconexión de redes de computadoras implementado en un conjunto de protocolos denominado TCP/IP y garantiza que redes físicas heterogéneas funcionen como una red (lógica) única.

Intranet: Una intranet es una red de ordenadores de una red de área local (LAN) privada empresarial o educativa que proporciona herramientas de Internet

JAVA: Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990. Las aplicaciones Java están típicamente compiladas en un "bytecode", aunque la compilación en código máquina nativo también es posible.

KPI: Key Performance Indicators (KPI) son métricas utilizadas para cuantificar objetivos que reflejan rendimiento estratégico de una organización.

Mainframe: Una computadora central o "mainframe" es una computadora grande, potente y costosa usada principalmente por una gran compañía para el procesamiento

de una gran cantidad de datos; por ejemplo, para el procesamiento de transacciones bancarias.

Máquinas virtuales: Una Máquina Virtual de Java (JVM) es el elemento encargado de ejecutar el código (bytecode) generado por la compilación de código fuente escrito usando el lenguaje de programación Java. Como todas las piezas del rompecabezas Java, fue desarrollado originalmente por Sun Microsystems.

Microsoft: Es una empresa multinacional americana fundada en 1975 dedicada al sector de las tecnologías de la información con sede en Redmond, Washington, Estados Unidos. Es mayormente conocida por sus sistemas operativos Windows y aplicaciones ofimáticas como Microsoft Office.

Middleware: El "Middleware" es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red).

Módulo: Un módulo es un componente autocontrolado de un sistema, el cual posee una interfaz bien definida hacia otros componentes; algo es modular si es construido de manera tal que se facilite su ensamblaje, acomodamiento flexible y reparación de sus componentes.

MQSeries: Es un Middleware de la familia de los softwares de IBM.

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 e-business.

Open source: Código abierto (del inglés open source) es el término con el que se conoce al software distribuido y desarrollado libremente. Fue a utilizado por primera vez en 1998 por algunos usuarios de la comunidad del software libre, tratando de usarlo como reemplazo al ambiguo nombre original en inglés del software libre (free software).

Patrones de diseño: Un Patrón de Diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).

Point-to-point: Point-to-point Protocol, es decir, Protocolo punto a punto, es un protocolo de nivel de enlace estandarizado en el documento RFC 1661. Por tanto, se trata de un protocolo asociado a la pila TCP/IP de uso en Internet. Más conocido por su acrónimo: PPP.

Proxies: En el contexto de las ciencias de la computación, el término "Proxy" hace referencia a un programa o dispositivo que realiza una acción en representación de otro. La finalidad más habitual es la del servidor proxy, que sirve para permitir el acceso a Internet a todos los equipos de una organización cuando sólo se puede disponer de un único equipo conectado, esto es, una única dirección IP.

REST: Representational State Transfer. Estilo arquitectónico propuesto por Roy Fielding en el 2000.

RMI: (Java Remote Method Invocation) es un mecanismo ofrecido en Java para invocar un método remotamente. Al ser RMI parte estándar del entorno de ejecución Java usarlo provee un mecanismo simple en una aplicación distribuida que solamente necesita comunicar servidores codificados para Java.

Repositorio: Un repositorio, depósito o archivo es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

RPC: El 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.

RUP: Proceso Unificado de Rational (RUP, en inglés Rational Unified Process) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

Smart Client: Es un término utilizado en el desarrollo de software, generalmente de aplicaciones web que no requieren de instalación, se actualizan automáticamente y tienen la apariencia de aplicaciones “desktop”.

SMTP: Simple Mail Transfer Protocol (SMTP), o protocolo simple de transferencia de correo electrónico. Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras o distintos dispositivos (PDA's, teléfonos móviles, etc.).

Software AG: Es una empresa alemana de software, muy conocida por su sistema gestor de bases de datos Adabas. Ha logrado un éxito enorme en el mercado con su solución orientada a servicios con la suite de integración Crossvision.

SQL: El Lenguaje de Consulta Estructurado (Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

Subtyping: El subtipado es un tipo de datos que generalmente esta relacionado a otro dato (súper dato) por una noción de sustitución, o sea que un programa puede operar con el dato y con elementos del súper dato

TCP: El Protocolo de Control de Transmisión (TCP en sus siglas en inglés, Transmission Control Protocol) es uno de los protocolos fundamentales en Internet. Muchos programas dentro de una red de datos compuesta por ordenadores pueden usar TCP para crear conexiones entre ellos a través de las cuales enviarse un flujo de datos.

UI: User Interface (Interfaz de usuario), la interfaz de usuario es la forma en que los usuarios pueden comunicarse con una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo.

UML: Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

W3C: Diferentes organizaciones, procedentes de diversos puntos del mundo y de campos muy diferentes, forman parte del W3C con intención de participar en un foro neutral para la creación de estándares Web.

Web 2.0: La Web 2.0 es la representación de la evolución de las aplicaciones tradicionales hacia aplicaciones web enfocadas al usuario final. El Web 2.0 es una actitud y no precisamente una tecnología.

WS-CDL: Este es un lenguaje utilizado para la definición de servicios dentro de la plataforma SOA, basado en XML y cuyo objetivo es la descripción del comportamiento de cada uno de los servicios establecidos para lograr un objetivo común.

WS-I: Web Services Interoperability Organization (WS-I), es un consorcio industrial caracterizado por promover la interoperabilidad entre la extensa gama de las especificaciones de los servicios web.

WS-PBEL: Es, en conjunto con WS-CDL, una alternativa para la implementación y manejo de Servicios Web. WS-BPEL surge como necesidad de ser el integrador o el engranaje para las diversas tecnologías que funcionan bajo SOA, pero que no logran una interoperabilidad al 100%, lo que restringe su funcionamiento y adaptabilidad. BPEL tiene como objetivo primordial la" orquestación" de servicios Web, es decir, la definición de un nuevo servicio Web a partir de servicios Web existentes.

WS-ReliableMessaging: Describe un protocolo que permite que los mensajes sean entregados relativamente entre aplicaciones distribuidas con la presencia de componentes de software, sistema, o fallas de red.

XML-RPC: Es un protocolo de llamada a procedimiento remoto que usa XML para codificar las llamadas (nivel de presentación) y HTTP como protocolo de aplicación.