UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 1

Dirección de Informatización.

TÍTULO: Propuesta de Arquitectura Orientada a Servicios para el Portal de Servicios Comunitarios**.**

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

Autores:

Yergeni Cabranes Pérez

Raly Martínez Porra

Tutor:

Yohana Baro Montenegro. Ingeniera en Ciencias Informáticas.

Ciudad de la Habana. Junio 2008

A:

Mi madre, mi hermano, mi padre y mi tío Conti que Dios le ayude…

Yergeni.

A:

Mis padres que son mi razón de ser….mis abuelos, mi familia en general…y a mi tía Juany, donde quiera que estés.

Raly.

Agradecimientos

Agradezco a:

Mí querida madre por todo el apoyo y esfuerzo que dedica a mi bienestar.

Mi hermano que me ha ayudado tanto, gracias, ahora me toca a mí.

Mi padre que siempre se ha preocupado por mis estudios.

Yohana por ayudarnos en todo momento sobre todo cuando lo necesitamos.

Chony por ayudarnos con su conocimiento y disposición.

Leo, por su solidaridad y dedicación para con nosotros.

Costilla por ser más que mi amigo mi hermano.

Todos los que de alguna forma hicieron posible este trabajo y mi convivencia en la universidad.

Yergeni.

Agradecimientos

Quiero agradecer a:

Mi mamá y mi papá por estar siempre a mi lado...

Mi tutora Yohana por ser tan comprensible...

Todos los que colaboraron con Yergeni y conmigo...

Mis coleguillas de la UCI, que no los nombro porque son muchos...

En general, a todos los que siempre han creído en mí.

Raly.

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Dirección de Informatización de La Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____ _____

Yergeni Cabranes Pérez Raly Martínez Porra.

Opinión del tutor

Título: **Propuesta de Arquitectura Orientada a Servicios para el Portal de Servicios Comunitarios.**

Autor(es): Yergeni Cabranes Pérez y Raly Martínez Porra.

El tutor del presente Trabajo de Diploma considera que durante su ejecución el estudiante mostró las cualidades que a continuación se detallan.

Por todo lo anteriormente expresado considero que los estudiantes están aptos para ejercer como Ingenieros en Ciencias Informáticas; y propongo que se le otorgue al Trabajo de Diploma la calificación de ___.puntos.

____ días del mes de Mayo del año 2008.

_____

Ing. Yohana Baro Montenegro.

Resumen

La Universidad de las Ciencias Informáticas, recinto donde se concentra la mayor comunidad informática de Cuba, juega un papel importante en el desarrollo de la informatización de la sociedad cubana. Este centro no está ajeno al avance y el desarrollo de las nuevas tecnologías de la información, que conlleva a que los proyectos actualicen y evolucionen sus aplicaciones de software. Este proceso requiere mucho tiempo, esfuerzo y capital. La solución está en definir una arquitectura de software que permita acomodarse a los cambios minimizando las situaciones problemáticas. Actualmente existen muchos modelos de arquitecturas que pueden dar solución a esta situación; pero para esto se requiere estandarización, reutilización, accesibilidad e integración. Es en este punto donde el paradigma orientado a servicios ha logrado imponerse.

El presente trabajo cuyo titulo es: Propuesta de Arquitectura Orientada a Servicios para el Portal de Servicios Comunitarios, presenta una caracterización del sistema comunitario actual, así como los problemas que está presentando. Se realiza toda una investigación de los principales componentes de una arquitectura orientada a servicios y sus características. El siguiente estudio está estrictamente enfocado a la actual necesidad que presenta en estos momentos la universidad y el país, en cuanto al uso del software libre para el desarrollo de aplicaciones. De forma general el objetivo es proponer un modelo de arquitectura orientada a servicios que se ajuste a esta necesidad.

Este documento puede ser fuente de consulta para el desarrollo de sistemas con características similares, en diversos entornos.

Palabras claves.

Arquitectura de Software, SOA, Servicios web, UDDI, ESB, BPM.

Índice

Índice de figuras

Introducción

Actualmente el avance de la tecnología ha jugado un papel importante en el desarrollo de aplicaciones empresariales. No obstante esta tarea no ha sido fácil aún considerando este avance. Los desarrolladores de este tipo de aplicaciones continúan enfrentándose a problemáticas y desafíos tales como datos complejos, requerimientos de negocio no explícitos y cambiantes constantemente, plataformas heterogéneas e interdependencias entre aplicaciones distribuidas. La falta de estándares que optimicen la construcción de estas aplicaciones ha llevado a desarrollar modelos más costosos y que demoran más tiempo en desplegarse, a partir del principio de que las propias empresas necesitan buscar o desarrollar sus propios estándares.

En el mercado empresarial los negocios empresariales exigen cada vez crear aplicaciones complejas en menor tiempo y con el menor costo posible. Es por esta situación que el mundo empresarial se está enfocando a incrementar la flexibilidad de los servicios y a la vez simplificar la infraestructura tecnológica utilizando los servicios web.

Con el objetivo de asumir este enfoque, la Dirección de Informatización, rectora de la gran mayoría de los proyectos productivos que se desarrollan actualmente en la Universidad de las Ciencias Informáticas. Dentro de las políticas que delega y que deben ser definidas en los proyectos que dirige, se encuentra la de construir una arquitectura orientada a servicios (SOA), así como la migración a software libre de todos los sistemas de gestión que permiten la informatización de los procesos en la universidad.

Situación Problémica:

Actualmente Servicios Comunitarios cuenta con una arquitectura de 3 capas. Es una arquitectura que depende de un servidor de aplicaciones, lo que la hace menos disponible y fiable. Cuenta con una mayor complicación arquitectónica, necesitando así de más software y hardware para su funcionamiento, haciéndola más costosa y menos rentable. Se basa

principalmente en la integración punto a punto que tiene muchos conflictos con direcciones de IP y no es una arquitectura que adopta rápidamente los cambios y las demandas del cliente.

Adoptando una Arquitectura Orientada a Servicios se obtendrá un menor coste de mantenimiento, una mayor eficiencia en la adaptación, capacidad de respuesta y despliegue que respondan a las demandas de los clientes.

Problema Científico:

¿Qué modelo de Arquitectura Orientado a Servicios se le puede definir al Portal de Servicios Comunitarios para el desarrollo e interoperabilidad de sus servicios?

Objetivo General:

Proponer un modelo de Arquitectura Orientado a Servicios para lograr el desarrollo y la integración del portal de Servicios Comunitarios.

Objetivos Específicos:

Definir un modelo de Arquitectura Orientado a Servicios que permita la interoperabilidad de los servicios que ofrece el portal.

Posibles resultados:

Un modelo de Arquitectura Orientado a Servicios basada en software libre para el desarrollo y la interoperabilidad de los servicios del Portal.

Objeto de estudio:

Las Arquitecturas Orientadas a Servicios existentes para el desarrollo de aplicaciones y sus componentes en cuanto a la optimización y adaptación a nuestras necesidades.

Campo de acción:

Portal de Servicios Comunitarios.

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

- ¿Cuáles son los fundamentos de las principales arquitecturas usadas actualmente?
- ¿Cuáles son las herramientas de software utilizadas por las principales arquitecturas?
- ¿Cómo diseñar un modelo de arquitectura acorde con las necesidades de la universidad?

Tareas de Investigación:

- Estudiar e investigar el estado del arte de la arquitectura existente en el proyecto.
- Estudiar los diferentes tipos de arquitecturas que existen actualmente en el ámbito internacional para lograr hacer un trabajo con calidad.
- Investigar acerca de las plataformas utilizadas para la implementación de soluciones.
- Estudiar los principales componentes y la estructura de una Arquitectura Orientado a Servicios.
- Definir servicios comunes para la interoperabilidad del portal.
- Proponer un Bus de Servicio Empresarial (ESB) y un Administrador de Procesos de Negocio (BPM) open source.
- Proponer un modelo de Arquitectura Orientado a Servicios acorde con las condiciones de La Universidad de Las Ciencias Informáticas.

En el trabajo hacemos uso de varios **métodos científicos de investigación** como son:

Histórico-Lógico referente a los de tipo teóricos, con el objetivo de hacer una investigación lo más detallada posible del surgimiento de la arquitectura de software, las principales características de los diferentes tipos de arquitecturas así como conceptualizaciones importantes para los próximos capítulos.

Modelación con el objetivo de realizar una abstracción y modelaje de la propuesta de arquitectura y sus componentes, así como la relación que va a existir entre estos.

Sistémico para describir la relación que se establecerá entre los componentes de la arquitectura. Se utilizará para estructurar y establecer dependencias entre dichos componentes, evaluar la calidad de la arquitectura en general y determinar posibles errores y resultados.

Experimental referente a los de tipo empíricos con el objetivo de establecer comparaciones entre la arquitectura existente y la propuesta que se realizará.

Análisis económico:

El proyecto Servicios Comunitarios actualmente cuenta con 40 estudiantes, entre ellos 13 de 5to año, 12 de 4to año, 5 de 3re año y 10 de 2do año y 4 profesores que se les paga un estipendio y salarios respectivamente, por aproximadamente 3 años. Se usan 11 computadoras para el desarrollo del proyecto, dichas PC son material de la UCI en las cuales se analizará su depreciación por el uso y el gasto de energía eléctrica. Realizando un estudio de cada factor se determinó que el costo general del proyecto es de $214 200 aproximadamente.

Este trabajo está compuesto por tres capítulos:

- Capítulo 1: Estado del arte. Fundamentación Teórica.

En este capítulo se estudiará la evolución de la arquitectura de software, así como los principales tipos de arquitecturas usadas mundialmente. Se definirá una propuesta de arquitectura con un estudio minucioso de sus principales características, definiciones y ventajas.

- Capítulo 2:

En este capítulo se abordará la plataforma web libre y el framework de desarrollo que se utilizará, así como un estudio de los principales aspectos de una SOA. Se propondrá un ciclo de vida para la propuesta y se realizará un estudio de los principales componentes de la misma.

- Capítulo 3:

En este capítulo se definirán e identificarán los servicios más comunes para el portal y se establecerá el medio de comunicación. Se hará un estudio más profundo de los componentes de una SOA, se propondrá un ESB y un BPM, ambos open source y se estructurará y presentará una propuesta de un modelo de la arquitectura SOA.

Capítulo 1

Fundamentación teórica. Estado del arte

1.1 Introducción.

En este capítulo se abordaran temas vinculados al desarrollo de la arquitectura de software, así como los diferentes tipos de arquitecturas mas usadas en el mundo, se estudiarán sus características, ventajas y desventajas. Se expondrá además un estudio de la arquitectura existente actualmente en el proyecto y un estudio de la propuesta.

1.2 Evolución de La Arquitectura de Software:

En los inicios de la informática, la programación se consideraba un arte, debido a la dificultad que representaba para la mayoría de las personas, pero con el tiempo se han ido desarrollando metodologías y fórmulas para conseguir los diferentes propósitos y hacer menos compleja la programación. A todas estas técnicas se les ha llamado Arquitectura de Software, un término que desde los años 60’ se ha estado manejando y desde entonces a ayudado a resolver infinidades de problemas.

No importa la envergadura, los requerimientos, ni el impacto social que tenga un sistema informático todos de alguna forma está estructurado por una arquitectura, algunas más simples y otras más complejas. La arquitectura ha permitido estudiar sistemas que ya estaban implementados y desarrollar otros nuevos, según la categoría del problema que resuelven y el tipo de arquitectura de software que se emplean para realizarlos, además de reducir el tiempo de solución, los costos y gastos de operación a significativos rasgos.

Uno de los principios de las metodologías modernas de desarrollo de software es priorizar la definición, el diseño, la implementación y la evaluación de la arquitectura del software, que es como se conoce al esqueleto de soporte del sistema. La arquitectura implementa los requisitos más críticos a través de las estructuras de programa de mayor alcance en el sistema. Por ello la arquitectura encierra los mayores riesgos del desarrollo. La clave del éxito

y, a su vez, la dificultad del principio de priorizar la arquitectura consiste en definir qué es y qué no es la arquitectura de software. **(Mollineda, 2005)**

1.3 ¿Qué es la arquitectura de software?

Actualmente en el mundo se han expuesto infinidades de definiciones de arquitectura de software. Aunque sean múltiples las definiciones todas se basan en tres aspectos fundamentales.

1. El trabajo dinámico de estipulación de la arquitectura dentro del proceso de ingeniería o el diseño (su lugar en el ciclo de vida).
2. La configuración o topología estática de sistemas de software contemplada desde un elevado nivel de abstracción.
3. La caracterización de la disciplina que se ocupa de uno de los dos asuntos anteriores, o de ambos.

La definición “oficial” de arquitectura de software se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000, adoptada también por Microsoft, que reza así:

"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución". **(Bylli Reynoso, 2006)**

Otra definición muy completa y reconocida es la de Clements[1], que plantea:

“La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan

[1] Dr. Paul C. Clements, trabajó para el Naval Resarch Laboratory en EE.UU. Washington. Allí creo una metodología en tiempo real para el desarrollo de software embebida, en sistemas con ciclos de vida largos, diseñados e implementados para software de aviación. Publicó libros y artículos de gran envergadura vinculados al desarrollo del software.

para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones." **(Bylli Reynoso, 2006)**

Ante el número y la variedades de definiciones existentes de arquitectura de software. También se ha definido en función de distintas clases de modelos, estos son:

Modelos estructurales: Sostienen que la arquitectura de software está compuesta por componentes, conexiones entre ellos y (usualmente) otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos y necesidades de los participantes. Caracterizado por el desarrollo de lenguajes de arquitecturas.
Modelos de framework: Son similares a la vista estructural, pero su énfasis primario radica en la (usualmente una sola) estructura coherente del sistema completo, en vez de concentrarse en su composición.
Modelos dinámicos: Enfatizan la cualidad conductual de los sistemas. "Dinámico" puede referirse a los cambios en la configuración del sistema, o a la dinámica involucrada en el progreso de la computación, tales como valores cambiantes de datos.
Modelos de proceso: Se concentran en la construcción de la arquitectura, y en los pasos o procesos involucrados en esa construcción. En esta perspectiva, la arquitectura es el resultado de seguir un argumento (script) de proceso.

Existen muchas definiciones de arquitecturas de software y aun ninguna de ellas ha sido como tal, totalmente aceptada. De forma general arquitectura de software se podría concluir que es el diseño de más alto nivel de un sistema, programa o aplicación y tiene la responsabilidad de:

1. Definir los módulos principales, sus responsabilidades y la interoperabilidad que existe entre ellos.
2. Control y flujo de datos.
3. Secuenciación de la información.
4. Protocolos de interacción y comunicación.

5. Ubicación del hardware.

Lo que no es arquitectura de software.

La arquitectura de software no es:

- Una normativa madura.
- Igual en la academia y la industria.
- Diseño de Software con UML.
- Naturalmente vinculada a metodología (RUP).
- Naturalmente relacionada con modelado Orientado a Objetos.
- Un vínculo “natural” entre requerimientos (casos de uso) y clases.

1.4 Arquitectura actual.

Servicios Comunitarios actualmente cuenta con 2 grupos de módulos, los referentes a servicios y los de reservación de transporte.

Módulos de Servicios.

1. Sistema Financiero Presupuestario.
2. Sistema para la gestión de los procesos de gas.

Módulos de reservación de trasporte.

1. Transporte Estudiantil.
2. Transporte Nacional.
3. Transporte de Profesores.
4. Trompo.

El modelo de arquitectura es de 3 capas con sus respectivas capas de presentación, lógica de negocio y acceso a datos. Para la presentación se definió los estándares web HTML, AJAX y JavaScript necesarios para realizar los sitios web. Se utiliza lenguaje PHP en la capa de

negocio, con el servidor web Apache y para el acceso a datos se utiliza el ORM (Mapeado Objeto Relacional) PROPEL.

Se utiliza sistema operativo Windows y GNU/Linux para analistas y programadores respectivamente. Para el modelado lenguaje UML y herramienta de modelado Visual Paradigm for UML Enterprise Edition. Para desarrollar los servicios web y toda la lógica de negocio se utiliza herramientas de lenguaje en PHP 5 como Zend Studio y Macromedia DreamweaverMX y el framework CakePHP como estándar de programación. Para las bases de datos se utiliza el gestor de bases de datos PostgreSQL.

1.5 Usabilidad en Arquitectura de Software.

Hasta hace poco, se asumía que la usabilidad era una propiedad exclusiva de la presentación de la información. Se creía que, encapsulando la capa de presentación y separándola del resto, se podía desarrollar la aplicación y, de forma iterativa, pasar los test de usabilidad. Tras cada test, tan sólo sería necesario resolver los problemas modificando la presentación y, gracias a esta separación, la funcionalidad no quedaría afectada.

Es recomendable relacionar el modelo del usuario con el modelo de objetos propio de la interfaz de usuario, en el sentido estricto de La Programación Orientada a Objetos. Este modelo de objetos, es el que permite al usuario relacionar sus objetivos con la funcionalidad del sistema. Por lo tanto para conseguir una buena usabilidad, no basta con tener en cuenta la capa de presentación, sino que es preciso que la usabilidad se contemple también en el momento de la definición de la funcionalidad de la aplicación.

Sin embargo no basta con tener en cuenta la presentación y la funcionalidad. Sobre todo en sistemas complejos, aquéllos 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.

La usabilidad busca entre otras cosas que el usuario pueda visualizar el progreso de sus peticiones, que pueda deshacer acciones, que pueda disponer de un entorno multilingüe, que

pueda cancelar una operación que lleva mucho tiempo en espera, que pueda reutilizar información que ha introducido anteriormente, y muchas otras cosas. Si analizamos estos escenarios de interacción, veremos que 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. **(Casanovas, 2004)**

De forma general las principales técnicas de la usabilidad son:

1. Autonomía, los usuarios deben tener el control sobre el sitio web.
2. El usuario manda, sería semejante a la regla del cliente siempre tiene la razón.
3. En la web calidad es igual a rapidez. (más de 20 segundos el 80% de los visitantes se irá sin ver la página)
4. Si el sitio no carga enseguida, debe ponerse una precarga, así el usuario permanece en el sitio.
5. El sitio debe ser sencillo, para que los usuarios se sientan cómodos y no se pierdan cada vez que necesiten encontrar algo en la página Web.
6. Poner breves conclusiones al principio ayuda al usuario a encontrar lo que busca en poco tiempo.
7. Escriba los contenidos resumidos un 25% de lo que pondrías en un papel (leer en pantalla cuesta mucho)
8. No sobrecargar de información a los usuarios, deben realizarse recorridos de lectura, quedando a la vista cual es la información principal y cual la secundaria.
9. Los colores han de utilizarse con precaución para no dificultar el acceso a los usuarios daltónicos (aprox. un 15% del total).
10. Utilizar interfaces conocidas, los sitios web deben requerir un mínimo proceso de aprendizaje.
11. Legibilidad, el color de los textos debe contrastar con el del fondo, y el tamaño de fuente debe ser suficientemente grande.

1.6 Tipos de Arquitecturas.

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. Consiste básicamente en que un programa (cliente informático) realiza peticiones a otro programa (el servidor).
Arquitectura de tres capas. Especialización de la arquitectura cliente-servidor donde la carga se divide en tres partes (o capas) con un reparto claro de funciones: una capa para la presentación (interfaz de usuario), otra para el cálculo (donde se encuentra modelado el negocio) y otra para el almacenamiento (persistencia). Una capa solamente tiene relación con la siguiente.
Arquitectura Orientada a Servicios (SOA)[2]: Define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

Existen otras arquitecturas menos usadas o menos conocidas, estas son:

1. Arquitectura en Pipeline.
2. Arquitectura entre Pares.
3. Arquitectura en Pizarra.

---

[2] En muchos sitios en internet esta arquitectura es abordada como menos conocida o menos usada, a pesar de esto nuestra investigación nos ha demostrado que actualmente es una de los modelos más usados por las grandes empresas como Microsoft, IBM y Software AG. Es por eso que la ponemos dentro de las más conocidas para nuestro entender.

1.6.1 *Arquitectura monolítica*. *Problemas.*

Se presentan en sistemas en que el mismo programa se encarga del manejo del sistema a todos los niveles. El mismo programa controla todo el proceso, desde las aplicaciones más bajas del sistema hasta la interface gráfica al usuario.

Hoy en día existen estándares y protocolos que pueden asegurar una buena relación entre las partes de la arquitectura. Un sistema que maneje todo es un programa muy grande que requiere de más capacidad de proceso y obliga al usuario a estar familiarizado con todos los elementos. Un sistema modular requiere que el usuario conozca solamente el módulo que va a usar. **(Orlich, 1998)**

Principales problemas:

1. Cuellos de botellas:

- 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 usuarios el servidor se satura.
- Si la visualización es compleja y necesita refresco inmediato la red se satura.

2. Fiabilidad:

- Si el sistema central falla, falla todo, incluso las aplicaciones que no usan datos y las que no son usadas 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, como ordenadores locales con datos en local, sistemas de base de datos monopuesto o monousuario, entre otras.

1.6.2 *Arquitectura Clientes-Servidor. Ventajas y desventajas.*

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

- 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.
- Es más flexible que el paradigma del P2P para poner al día los datos u otros recursos.
- Hay tecnologías maduradas diseñadas para el paradigma Cliente/Servidor(C/S) que ofrecen seguridad, la facilidad de la interfaz, y la facilidad de empleo.

El servidor de cliente es la arquitectura de red que separa al cliente (a menudo un uso que utiliza un interfaz utilizador gráfico) de un servidor. Cada caso del software del cliente puede enviar peticiones a un servidor. Los tipos específicos de servidores incluyen los servidores web, los servidores del uso, los servidores de archivo, los servidores terminales, y los servidores del correo. Mientras que sus propósitos varían algo, la arquitectura básica sigue siendo igual.

Desventajas.

- La congestión del tráfico ha sido siempre un problema desde el primer día del nacimiento del paradigma de C/S. Cuando una gran cantidad de clientes envían peticiones al mismo servidor al mismo tiempo, puede ser que cause muchos de los apuros para el servidor. Más clientes hay más apuros que tiene. Mientras que, el ancho de banda de la red del P2P se compone de cada nodo en la red, cuanto más nodos hay, mejor el ancho de banda que tiene.
- El paradigma de C/S no tiene la buena robustez como red del P2P. Cuando el servidor está caído, las peticiones de los clientes no pueden ser satisfechas. En la mayor parte de redes del P2P, los recursos están situados generalmente en nodos por todas partes de la red. Aunque algún nodos salen o abandonan la descarga; otros nodos pueden todavía acabar de descargar consiguiendo datos del resto de los nodos en la red.
- El software y el hardware de un servidor es generalmente muy terminantes. Un hardware regular del ordenador personal puede no poder servir sobre cierta cantidad de clientes. Mientras tanto, una edición casera de Windows XP incluso no tiene IIS a trabajar como servidor. Necesita software y el hardware específico satisfacer el trabajo. Por supuesto, aumentará el coste.

1.6.3 *Arquitectura de tres capas. Ventajas. Capas y 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 de esto 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 solo se ataca al nivel requerido sin tener que revisar entre código mezclado. 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.

Las capas son las siguientes:

Capa de presentación: es la que ve el usuario (hay quien la denomina "capa de 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, se reciben las peticiones del usuario y se envían 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 y es la encargada de acceder a los datos. Está formada por uno o más gestores de bases de datos que realizan 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 es lo típico). Si bien lo más usual es que haya una multitud de ordenadores en 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 mas 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.

Seria recomendable especificar que 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). El término "nivel", corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física (ordenadores).

1.6.4 *Arquitectura orientada a servicios (SOA). Propuesta*

En la actualidad el impacto en las tecnologías de la información (IT) y los procesos de negocios que ha tenido el surgimiento de SOA (Arquitectura Orientada a Servicios) es revolucionario. Este modelo de arquitectura permite que las organizaciones combinen fluidamente sus activos de tecnologías de información existentes para desarrollar nuevas aplicaciones, procesos y modelos de negocios internos o externos a la empresa, de una forma más económica rápida y sencilla. Actualmente en el mundo existen grandes empresas que han utilizado SOA para soluciones de mercado con magníficos resultados. Son los casos de Microsoft, IBM, SUN Microsystems, BEA Systems, Software AG, entre otras.

SOA como tal empieza a sonar en el mercado en el año 2000 y son muchas las definiciones echas hasta ahora para este modelo. Citamos:

Según W3C: “Conjunto de componentes que pueden ser invocados, cuyas descripciones de interfaces se pueden publicar y descubrir”. (SOA Introduction, 2006)

Según CBDI: “Estilo resultante de políticas, prácticas y frameworks que permiten que la funcionalidad de una aplicación se pueda proveer y consumir como conjuntos de servicios, con una granularidad relevante para el consumidor…” (SOA Introduction, 2006)

El objetivo de SOA es aportar para lograr una alta performance desarrollando servicios que puedan ser reutilizados por la organización. Estas capacidades permiten que las tecnologías respondan rápidamente a los cambios de los negocios, los escenarios competitivos, las alianzas y las necesidades de los consumidores. SOA permite la creación de aplicaciones compuestas que trabajan a través de una única interfaz, facilitando la comprensión de distintos procesos de negocios. (Simoes)

Características y ventajas de una SOA.

Una SOA define las siguientes capas de software:

Aplicativa básica: Sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad.

Exposición de funcionalidades: Donde las funcionalidades de la capa aplicativas son expuestas en forma de servicios (webservices).

Integración de servicios: Facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración.

Composición de procesos: Define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio; de entrega, donde los servicios son desplegados a los usuarios finales. Además al contrario de las arquitecturas orientada a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables.

SOA presenta grandes ventajas en su adopción por parte de las empresas:

- Mejora en los tiempos de realización de cambios en procesos.
- Facilidad para evolucionar a modelos de negocios basados en tercerización.
- Facilidad para abordar modelos de negocios basados en colaboración con otros entes (socios, proveedores).
- Poder para reemplazar elementos de la capa aplicativa SOA sin disrupción en el proceso de negocio.
- 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.

Diseño y desarrollo con SOA

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 se refieren a 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.

SOA y el software libre.

Una SOA no es un estilo de arquitectura que impulsa algún lenguaje en particular, sistema operativo o tecnología. ¿Será confiable y aplicable la utilización de Software Libre en un ambiente SOA?

Más del 60% de internet utiliza software libre, existen numerosos proyectos con mucha inversión y aportes de corporaciones que entienden que el software libre es sustentable, y porque el software libre existe desde hace más de 20 años y no ha parado de crecer desde entonces, liberando más y más aplicaciones cada año.

Dentro de los principios de SOA se encuentran la reutilización, la interoperabilidad y el uso de estándares (entre otros). Estos principios se encuentran en la mayoría del software libre.

Reutilización: Este modo de crear tecnología impulsa que el código se pueda reutilizar, ya sea porque la licencia que gobierna el software exige que el código no pueda ser cerrado y

pueda cambiarse, siempre y en cuanto la nueva aplicación mantenga las mismas libertades, o porque permite que el software sí pueda ser cerrado y vendido como software propio.

Es muy común entre los programadores que liberan software el pensamiento de no reinventar la rueda, de reutilizar las cosas que funcionan bien, por lo que existen más garantías de que si algún componente de una arquitectura SOA es software libre entonces ese componente en particular haya sido probado por una gran cantidad de usuarios, programadores y administradores, por consiguiente sea más sólido, estable y libre de errores. Claro que dependerá enormemente de dónde se haya obtenido el código o componente.

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.

Aún hoy, existiendo mucho software libre, la interoperabilidad sigue siendo algo 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: La interoperabilidad en muchos casos se encuentra a través del buen uso de estándares, otro de los principios de SOA. En el inicio de cada proyecto que debe que interoperar necesariamente, primero se realiza la implementación de un estándar que sirva a los propósitos del software, luego, se hacen adaptaciones para poder interoperar con aplicaciones que no implementan perfectamente el estándar.

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. **(Coletti, 2006)**

¿Como se resuelven los retos de SOA?

Embarcarse en un proyecto de SOA supone tener que resolver una serie de retos, tanto a nivel organizativo como técnico, y estos retos pueden convertirse en verdaderas barreras insuperables. Para que las iniciativas de adopción de SOA tengan un fin satisfactorio, hay que asegurarse de que se cumplen una serie de condiciones indispensables:

- **Definir claramente los objetivos de negocio.** Disponer de una visión y un rumbo claro desde el principio hará mucho más fácil la ejecución de procesos cuya esencia es la integración de múltiples funciones.
- **Definir claramente el alcance del proyecto.** El objetivo real de cada iniciativa SOA debe ser responder a necesidades concretas de negocio y crear soluciones en pasos discretos, incrementales e iterativos.
- **Evitar introducir SOA sin motivos reales que lo justifiquen.** La adopción de SOA no debe considerarse una necesidad tecnológica, sino organizativa. Debe responder a las necesidades de la organización o del proyecto en concreto.
- **Gestionar procesos.** Los servicios y aplicaciones se corresponden con procesos y los outputs de información deseados a través de las diversas áreas funcionales de la organización. Puesto que representan procesos compartidos, es necesario que se les asigne un propietario para que puedan inventariarse y gestionarse a fin de garantizar que cumplen en todo momento con las directivas corporativas y responden adecuadamente a las necesidades que los justifican.

¿Como lograr una SOA?

Para lograr SOA, primero se deben exponer los recursos de TI existentes como servicios. Es probable que muchos de ellos, y sobre todo los sistemas legacy, no soporten nativamente los servicios web (la forma más común de lograr SOA) y que vengan de diferentes plataformas y vendedores, por lo que los adaptadores son necesarios para proveer conectividad con estos recursos. Después se requiere integrar estos servicios en una aplicación, paso en donde se pueden realizar mejoras y automatizaciones a los procesos al rediseñarlos, documentarlos,

implementarlos, monitorearlos y cambiarlos continuamente. Pero estos no tienen valor para el negocio si no hay beneficios para los usuarios, por lo que el último paso es hacer llegar los servicios a los usuarios y esto se puede lograr a través de aplicaciones web tales como portales, aplicaciones de Office o incluso a través de dispositivos móviles. Finalmente deben tomarse en cuenta actividades de seguridad y administración.

Por otra parte si bien una adopción de SOA bien planificada y ejecutada puede mejorar la capacidad de respuesta de las organizaciones, no todos los esfuerzos de orientación a servicios han resultado satisfactorios. Los proyectos de SOA han tenido un éxito limitado cuando los desarrolladores los han intentado resolver de abajo arriba: implantar SOA por el gusto de tener SOA sin tener una referencia clara del contexto de negocio en el que debe desplegarse es un proyecto sin principios organizativos y sin rumbo. El resultado será una implementación caótica que no aportará beneficio alguno a la empresa.

Principales elementos de una SOA.

Los elementos básicos que conforman una SOA son:

1. Proveedores de servicios.
2. Consumidores de servicios.
3. Bus Empresarial de Servicios.

La integración de aplicaciones es el mayor reto en la actualidad para muchos negocios. Un “Enterprise Service Bus” (ESB) es probablemente la vía mas 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, basado en la mejor práctica en patrones de diseño para la integración de aplicaciones.

Un bus de servicios empresariales es una solución de integración distribuida, basada en los mensajes y en estándares abiertos. La función de un ESB es proporcionar una comunicación

fiable entre los distintos recursos tecnológicos tales como aplicaciones, plataformas y servicios, que están distribuidos en múltiples sistemas por toda la empresa. A medida que los departamentos de TI se centran cada vez más en el diseño de SOA para reducir los costes de desarrollo y para aumentar la agilidad del negocio, los ESB se están convirtiendo en un primer paso clave para el establecimiento de una SOA empresarial. Los ESB constituyen los cimientos de una SOA y pueden complementarse con capacidades de productividad adicionales, como la orquestación de servicios y los registros. **(TIBCO)**.

La elección.

Como bien describe el nombre del proyecto, Servicios Comunitarios está vinculado a ofrecer prestaciones de servicios a la comunidad universitaria de la UCI. Adoptando una SOA podríamos gestionar mejor los servicios, proporcionar una comunicación más eficiente entre ellos, evitar dependencias a través del bajo acoplamiento que brinda la orientación a servicios, controlar, gobernar y asegurar las prestaciones. Por otra parte si se define un ciclo de vida para la adopción de una SOA, se obtendrá una arquitectura robusta que podrá definir componentes y funcionalidades para la creación de un portal idóneo.

1.7 Modelos y vistas de arquitectura de software.

Toda arquitectura de 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. Esto es así porque todas las vistas deben ser coherentes entre sí, evidente dado que describen la misma cosa.

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:

1. La visión estática: describe qué componentes tiene la arquitectura.

2. La visión funcional: describe qué hace cada componente.
3. La visión dinámica: describe cómo se comportan los componentes a lo largo del tiempo y como interactúan entre sí.

Uno de los modelos más conocidos es el 4+1 de Philippe Kruchten[3], vinculado al Rational Unified Process (RUP), que define cuatro vistas diferentes más una de escenarios:

1. Vista lógica: describe el modelo de objetos.
2. Vista de proceso: muestra la concurrencia y sincronía de los procesos.
3. Vista física: muestra la ubicación del software en el hardware.
4. Vista de desarrollo: describe la organización del entorno de desarrollo.
5. 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.

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, etc. Estos lenguajes son apropiados únicamente para un modelo o vista. Afortunadamente existe cierto consenso en adoptar UML (Unified Modeling Language, Lenguaje Unificado de Modelado) como lenguaje único para todos los modelos o vistas. Sin embargo, un lenguaje generalista corre el peligro de no ser capaz de describir determinadas restricciones de un sistema de información. (Garzás)

1.8 Estilos y patrones de Arquitectura de software.

Los estilos afectan a toda la arquitectura de software y puede combinarse en la propuesta de solución. De forma general las funcionalidades y objetivos para la arquitectura son:

---

[3] Philippe Kruchten: fue anteriormente el Director del Proceso de Desarrollo (RUP) en Rational Software. Actualmente es profesor de Ingeniería de Software en la Universidad de British Columbia en Vancouver, Canadá. Él diseñó el modelo 4 + 1 vista.

- Su utilización para sintetizar estructuras de soluciones.
- Encapsular una enorme variedad de configuraciones concretas.
- Definir los posibles patrones de las aplicaciones.
- Permitir evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

En cambio los patrones se enfocan a dar solución a un problema en específico, de un atributo de calidad, y abarca solo parte de la arquitectura. Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puedes usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces.

1.9 Conclusiones.

En este capítulo se realizó un estudio y una conceptualización de los principales aspectos del surgimiento de La Arquitectura de Software, así como las arquitecturas más usadas actualmente en el ámbito internacional. Se presentó una propuesta y un estudio muy conceptual de sus aspectos más relevantes.

Capítulo 2

Componiendo una SOA

2.1 Introducción:

En este capítulo se abordará la plataforma web libre y el framework de desarrollo que se utilizará, así como un estudio de los principales aspectos de una SOA. Se propondrá un ciclo de vida para la propuesta y se realizará un estudio de los principales componentes de la misma.

2.2 Plataformas web libres (PWL).

Para aplicaciones web existen varias plataformas y lenguajes de desarrollo en software libre. Quizás la más indicada para esta propuesta sea LAMP, plataforma 100% libre que se explica a continuación.

2.2.1 *Plataforma Linux, Apache, MySQL, PHP (LAMP).*

LAMP está considerada como una de las mejores herramientas disponibles para que cualquier organización o individuo pueda emplear un servidor web versátil y potente. Aunque creados por separado, cada una de las tecnologías que lo forman dispone de una serie de características comunes. Especialmente interesante es el hecho que estos cuatro productos pueden funcionar en una amplia gama de hardware, con requerimientos relativamente pequeños sin perder estabilidad. Esto ha convertido a LAMP en la alternativa más adecuada para pequeñas y medianas empresas. Independientemente del hecho que no es necesario pagar licencias por su utilización, su mantenimiento se reduce a actualizar paquetes que se pueden descargar por Internet y su nivel de seguridad es muy bueno, al liberarse parches de seguridad al muy poco tiempo que se declara una alerta.

Existen, no obstante, multitud de variaciones de código libre. La L de Linux puede ser sustituida por FreeBSD, NetBSD u OpenBSD. En lugar de la M de MySQL también podemos encontrar PostgreSQL. La P sirve para PHP, Perl, Python, y Ruby.

La "L" cubre la capa del sistema operativo con ***Linux***. En la actualidad el Linux se ha posicionado fuertemente en el sector de los servidores y está siendo utilizado cada día más para soluciones de misión crítica. Inclusive se conoce de casos de reemplazo de granjas de cientos de servidores por un mainframe corriendo bajo Linux, sin inconvenientes. En este punto proponemos el uso de dos sistemas operativos basados en GNU/Linux.

Debian (versión 3.1) o más exactamente, Debian GNU/Linux **(Debian, 2008)** es una distribución GNU/Linux que basa sus principios y fin en el software libre. Debian no vende directamente su software, lo pone a disposición de cualquiera en Internet, aunque sí permite a personas o empresas distribuir comercialmente este software mientras se respete su licencia. Algunas características de Debian son:

- La disponibilidad en varias plataformas hardware. La versión 3.1a es compatible con 11 plataformas.
- Una amplia colección de software disponible. La versión 3.1a viene con unos 15490 paquetes de software|paquetes.
- Un grupo de herramientas para facilitar el proceso de instalación y actualización del software.
- Su compromiso con los principios y valores involucrados en el movimiento del Software Libre.
- No tiene marcado ningún entorno gráfico en especial, ya sea GNOME, KDE u otro.

Ubuntu (versión 8.04) es una distribución Linux que ofrece un sistema operativo predominantemente enfocado a ordenadores de escritorio aunque también proporciona soporte para servidores.

Basada en Debian GNU/Linux, Ubuntu concentra su objetivo en la facilidad de uso, la libertad en la restricción de uso, los lanzamientos regulares (cada 6 meses) y la facilidad en la instalación. (Ubuntu)

La "A" denota ***Apache*** (versión 2.2.6) como servidor Web. El servidor Web simplemente se encarga de servir páginas estilo Web hacia quien lo solicita, que por lo general es un PC o dispositivo con un visor de Web. Apache es el servidor web por excelencia, con algo mas de un 60% de los servidores de internet confiando en él. Entre sus características mas sobresalientes están:

- **Fiabilidad:** Alrededor del 90% de los servidores con más alta disponibilidad funcionan con Apache.
- **Gratuidad:** Apache es totalmente gratuito, y se distribuye bajo la licencia, Apache Software License, que permite la modificación del código.
- **Extensibilidad:** Se pueden añadir módulos para ampliar las ya de por si amplias capacidades de Apache. Hay una amplia variedad de módulos, que permiten desde generar contenido dinámico (con PHP, Java, Perl, Pyton), monitorizar el rendimiento del servidor, atender peticiones encriptados por SSL, hasta crear servidores virtuales por IP o por nombre (varias direcciones web son manejadas en un mismo servidor) y limitar el ancho de banda para cada uno de ellos. Dichos módulos incluso pueden ser creados por cualquier persona con conocimientos de programación.

La “M” se refiere al gestor de bases de datos ***MySQL***. Es un servidor de bases de datos relacionales muy rápido y robusto. Es software libre, publicado bajo la licencia GPL (GNU Public License). Este gestor se creó con la rapidez en mente, de modo que no tiene muchas de las características de los gestores comerciales más importantes, como Oracle, Sybase o SQL Server. No obstante, eso no ha impedido que sea el más indicado para aplicaciones que requieren muchas lecturas y pocas escrituras y no necesiten de características muy avanzadas, como es el caso de las aplicaciones web. MySQL está disponible además para un enorme número de sistemas operativos.

También en este punto como ya mencionamos anteriormente otro gestor de bases de datos que puede tomar lugar es ***PostgreSQL*** (versión 8.1.2). Está considerado ampliamente como el sistema gestor de bases de datos de código abierto más avanzado en el mundo. Ofrece muchas ventajas respecto a otros sistemas de bases de datos:

Instalación ilimitada: Con la instalación de PostgreSQL no puede ser demandado por incumplir o violar acuerdas de licencia, puesto que no hay costo asociado a la licencia del software.

Esto tiene varias ventajas adicionales:

- Modelos de negocios más rentables con instalaciones a gran escala.
- No existe la posibilidad de ser auditado para verificar cumplimiento de licencia en ningún momento.
- Flexibilidad para hacer investigación y desarrollo sin necesidad de incurrir en costos adicionales de licenciamiento.

Ahorros en costos de operación: Ha sido diseñado y creado para tener un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características, estabilidad y rendimiento. Además de esto, los programas de entrenamiento son reconocidamente mucho más costo-efectivos, manejables y prácticos en el mundo real que aquellos de los principales proveedores comerciales.

Estabilidad y confiabilidad legendarias: En contraste a muchos sistemas de bases de datos comerciales, es extremadamente común que compañías reporten que PostgreSQL nunca ha presentado caídas en varios años de operación de alta actividad. Ni una sola vez. Simplemente funciona.

Extensible: El código fuente está disponible para todos sin costo. Si algún equipo necesita extender o personalizar PostgreSQL de alguna manera, puede hacerlo con un mínimo esfuerzo, sin costos adicionales. Esto es complementado por la comunidad de profesionales y

entusiastas de PostgreSQL alrededor del mundo que también extienden PostgreSQL todos los días.

Multiplataforma: PostgreSQL está disponible en casi cualquier Unix (34 plataformas en la última versión estable), Incluso para Windows.

Diseñado para ambientes de alto volumen: PostgreSQL usa una estrategia de almacenamiento de filas llamada MVCC para conseguir una mejor respuesta en ambientes de grandes volúmenes. Los principales proveedores de sistemas de bases de datos comerciales usan también esta tecnología, por las mismas razones.

Finalmente la "P" se refiere a lenguajes de programación web como ***PHP***, ***Perl*** y ***Python***. De estos tres, el que hasta hace poco tenía la delantera y todavía presenta buena acogida entre programadores es el Perl. Normalmente permite la elaboración de programas de una manera sencilla, y permite la interacción y la manipulación de información. Recientemente, con el avance que ha tenido el PHP en su versión 4 y la reciente liberación de la versión 5, se considera que éste será el lenguaje de programación de las soluciones Web en la plataforma abierta. Tiene también la posibilidad de ser instalado en múltiples plataformas, lo que lo convierte en una buena elección para empresas que quieran desarrollar a bajo costo. El PHP se caracteriza en términos generales por proveer funciones de conexión con las bases de datos y funciones de presentación para que el resultado sea presentado al usuario en formato de página Web. La cantidad de aplicaciones ya disponibles en PHP es amplia. Inclusive, la mayoría de las versiones Web de sus proveedores de correo tienen aplicaciones elaboradas en PHP. Se utilizará PHP en su versión 5

Con los años han nacido muchos lenguajes de programación web como ASP, JSP, entre otros, seria recomendable ver la diferencia de PHP con respecto a estas alternativas.

1. Es software libre, lo que implica menores costes y servidores más baratos que otras alternativas, a la vez que el tiempo entre el hallazgo de un fallo y su resolución es más corto. Además, el volumen de código PHP libre es mucho mayor que en otras tecnologías, siendo superado por Perl, que es más antiguo. Esto permite construir sitios realmente

interesantes con sólo instalar scripts libres como PHP Nuke (weblog, comunidad o bitácora), osCommerce (comercio electrónico con capacidad multilingüe), eZ publish (sistema de gestión de contenidos), phpBB (foros de discusión) o phpMyAdmin (administración de base de datos MySQL).

2. Es muy rápido. Su integración con la base de datos MySQL, también veloz, le permite constituirse como una de las alternativas más atractivas para sitios de tamaño medio-bajo.
3. Su sintaxis está inspirada en C, ligeramente modificada para adaptarlo al entorno en el que trabaja, de modo que si estás familiarizado con esa sintaxis, PHP o JSP son las opciones más atractivas.
4. Su librería estándar es realmente amplia, lo que permite reducir los llamados "costes ocultos", uno de los principales defectos de ASP.
5. PHP es relativamente multiplataforma. Funciona en toda máquina que sea capaz de compilar su código, entre ellas diversos sistemas operativos para PC y diversos Unix. El código escrito en PHP en cualquier plataforma funciona exactamente igual en cualquier otra.
6. El acceso a las bases de datos de PHP es muy heterogéneo, pues dispone de un juego de funciones distinto por cada gestor.
7. PHP es suficientemente versátil y potente como para hacer tanto aplicaciones grandes que necesiten acceder a recursos a bajo nivel del sistema como pequeños scripts que envíen por correo electrónico un formulario rellenado por el usuario.
8. Como lenguaje, PHP padece ciertas carencias: no soporta polimorfismo ni tiene excepciones u otro sistema de errores aceptable.

Las soluciones que ofrecen las casas comerciales se encargan de hacer que los cuatro componentes que se han analizado: sistema operativo, servidor Web, base de datos y lenguaje de programación; funcionen sincronizada y adecuadamente dentro de un conjunto de equipos. La propuesta LAMP pretende ser la respuesta con software y lenguaje que hasta ahora ha dado buenos resultados en forma separada y que promete siempre y cuando se logre consolidar el soporte para cada una de las herramientas (base de datos y lenguajes en

especial) garantizar la facilidad de instalación de cada uno de los componentes para permitir una integración transparente.

Algunas de las ventajas que se obtienen de utilizar LAMP son:

- Soporte a gran cantidad de arquitecturas, como son Intel y compatibles, SPARC, Mips y PPC (Macintosh).
- Código relativamente sencillo y con pocos cambios de una plataforma a otra.
- Parches generados en poco tiempo después de encontrarse un agujero de seguridad.
- Actualizaciones del software vía Internet.
- Posibilidad de incrementar los servicios y funciones desde el código fuente.

Sin embargo, tenemos también una serie de desventajas que deben considerarse:

- Es muy distinto de Windows, lo que dificulta el trabajo a quienes estén acostumbrados a él.
- Las actualizaciones requieren en ocasiones tener conocimientos profundos del sistema.
- Configurar algunos servicios de red requiere de más tiempo que en Windows.
- Mayor coste del personal.

En el proyecto Servicios Comunitarios actualmente existe un grupo de desarrolladores bastante avanzado en las herramientas y lenguajes que conforman la plataforma LAMP, no siendo así el caso del sistema operativo. Se recomienda en este aspecto para años posteriores impartir cursos dirigidos a sistemas operativos Linux, principalmente en Debian y Ubuntu, para que posteriormente sea Linux el sistema operativo por defecto que se encuentre instalado en los ordenadores del proyecto.

Para el desarrollo de los servicios del portal se utilizará ***NuSOAP.***

NuSOAP es un kit de herramientas (ToolKit) para desarrollar Web Services bajo el lenguaje PHP. Está compuesto por una serie de clases que nos harán mucho más fácil el desarrollo de Web Services. Provee soporte para el desarrollo de clientes (aquellos que consumen los Web Services) y de servidores (aquellos que los proveen). NuSOAP está basado en SOAP 1.1,

WSDL 1.1 y HTTP 1.0/1.1. Además está en una fase madura de desarrollo, no necesita módulos adicionales y es muy fácil su instalación y uso.

Para agilizar la implementación utilizamos el framework ***CakePHP*** del cual se hará un estudio:

CakePHP es un framework para PHP de código abierto para el rápido desarrollo de aplicaciones. Es una estructura de librerías, clases y estructuras run-time para programadores que creaban aplicaciones web originalmente inspiradas en el framework Ruby on Rails.

CakePHP tiene varias características que hacen que sea una gran elección como framework para el desarrollo de aplicaciones rápidamente y con la menor cantidad de molestia. Aquí están presentadas algunas de estas características, sin un orden particular:

1. Comunidad activa y amistosa.
2. El licenciar flexible.
3. Compatibilidad con PHP 4 y PHP 5.
4. CRUD integrado para la interacción de la base de datos y consultas simplificadas.
5. Scaffolding para aplicaciones (AMB de prueba simple).
6. Arquitectura Modelo Vista Controlador (MVC).
7. Despachador de URLs con un buen aspecto.
8. Validación incorporada.
9. El templating rápido y flexible (PHP syntax, con los helpers).
10. Ver Helpers para AJAX, Javascript, HTML formularios.
11. Seguridad, sesión, y componentes que manejan peticiones.
12. Listas flexibles del control de acceso.
13. Desinfección de datos.
14. View Caching flexible.
15. Trabaja desde cualquier subdirectorio de la web, con poca o ninguna configuración de Apache.

En cuanto al modelado y diseño se propone como lenguaje de modelado ***UML*** (Lenguaje Unificado de Modelación) y ***Visual Paradigm for UML Enterprise Edition*** (versión 6.0) como herramienta de modelado***.***

Básicamente el UML es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir. UML entrega una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables.

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. **(Visual Paradigm International, 2007)**

2.3 Gobernabilidad y Ciclo de Vida de una SOA.

Normalmente, gobernabilidad no es solo un conjunto de tecnologías que te permiten llevar a cabo satisfactoriamente una arquitectura SOA. Suele ser un grupo de herramientas en conjunto con diversas metodologías, guías y buenas prácticas como: registro de servicios, políticas de gestión, de seguridad, ciclos de vida, repositorios de información de todo tipo asociada a los servicios, etc.

Cuando se habla de una SOA, se refiere a servicios y asociado a ello, tenemos que hablar de ciclo de vida. Normalmente se evidencian 4 fases en un ciclo de vida SOA: *Análisis, Diseño, Desarrollo, Medida (o Monitorización) y Gobierno*.

Análisis: Comprende normalmente la fase de análisis del proyecto objeto de la Arquitectura SOA. Es importante analizar no solo lo existente sino también lo que se va a construir. Es necesario identificar como está estructurado el proyecto, las aéreas existentes y como se encuentra definido. Es en este punto donde debemos definir un Plan de Gobernabilidad correcto y adecuado a las necesidades (que podrá, no obstante, ser modificado a futuro si encontramos dicha necesidad).

Diseño: Una vez que se ha identificado la oportunidad para construir una Arquitectura SOA debemos definir los distintos niveles que serán necesarios, las personas que van a estar involucrada y en que medida.

Desarrollo: Será el momento de poner en funcionamiento lo diseñado en las fases previas. Habrá que poner en funcionamiento las distintas políticas que aseguren la calidad de los desarrollos. En este punto es donde, a muy nivel técnico, deben registrarse los servicios y hacerse disponibles según el propio ciclo de vida de aceptación para el resto de usuarios. Aquí es donde debemos sacar todo el rendimiento a las herramientas tecnológicas que nos permitirán llevar a cabo una gobernabilidad de la arquitectura de manera correcta.

Monitorización: La monitorización y visión del funcionamiento de la arquitectura nos permitirá determinar los fallos o carencias que tiene, y ofrecer una remodelación de los diseños iníciales. El cumplimiento de los niveles establecidos nos permitirá ver que se está haciendo bien y que se está haciendo mal. Este punto es en el cual se cierra el ciclo de vida y se permite el aseguramiento de la calidad por el refuerzo de la misma.

Gobierno: Esta fase es crítica para cualquier proyecto SOA. Para asegurar el éxito se puede crear un centro de excelencia dentro del negocio para implementar las políticas de gobierno y seguir comprobados estándares de control de información y tecnologías relacionadas.

La Gobernabilidad en SOA es un tema importante debido principalmente a la interdependencia de los sistemas involucrados: existen múltiples propietarios, múltiples políticas, etc. y ello conlleva generalmente que cada “incidencia” de reutilización pueda provocar dependencias adicionales.

2.4 Componentes de una SOA.

Una SOA al igual que las demás arquitecturas está formada por componentes necesarios para su funcionamiento y el procesamiento de información. El principal componente de una SOA son sin duda los servicios. De aquí que surjan interrogantes como: ¿Cuáles son los principios de la orientación a servicios?, ¿Cómo se definen los servicios?, ¿Qué tipos de servicios existen?, ¿Qué características debe cumplir un servicio? En este punto se estará abordando acerca de estas interrogantes.

2.4.1 *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. Estos principios 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 funcionales 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 y coreografía.
- **Los Servicios deben de ser autónomos:** Todo Servicio debe tener su propio entorno de ejecución. De esta manera el servicio es totalmente independiente y nos podemos 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.

Cuando se desarrollan aplicaciones SOA es muy útil y necesario tener en cuenta siempre estos principios, ya que nos van a dar las pautas necesarias para tomar ciertas decisiones de diseño complejas.

Como se abordó en el capítulo 1 en el punto 1.6.4 referente a SOA, existen un consumidor y un proveedor de servicios. Pues bien en este caso el proveedor es el encargado de desarrollar el servicio para que el consumidor lo utilice. Existe lo que se llama Lenguaje de Descripción de Servicios Web (WSDL por sus siglas en ingles). El WSDL no es un documento obligatorio, pero es muy importante que sea estándar ya que así se podrá acceder de manera dinámica a los servicios.

2.4.2 *WSDL (*Web Service Description Language*)*

Un Lenguaje de Descripción de Servicios Web no es más que un lenguaje basado en XML para describir el comportamiento de un servicio web (su API). Contiene las operaciones y los tipos de datos necesarios para definir dichas operaciones. Todo servicio web debe proveer su WSDL para que otros puedan invocarlo.

Es muy importante entender WSDL porque es la parte fundamental para desarrollar servicios. Quizás no sea necesario saber construir un documento WSDL (ya que lo construyen automáticamente las herramientas de desarrollo), pero sí entenderlo. Actualmente se encuentra la versión 2.0 de WSDL, pero se utilizará la versión 1.1 debido a que la nueva no es compatible con muchos servidores.

WSDL es un lenguaje basado en XML creado para definir la interfaz de los servicios. Un documento[4] WSDL está divido en dos partes claramente diferenciadas [ver figura 1]:

- **Parte concreta:** Es la parte que define el "como" y "donde".
- **Parte abstracta:** Es la parte que define qué hace el servicio a través de los mensajes que envía y recibe.

En la parte abstracta tenemos:

- **types:** Esta etiqueta define las estructuras de datos que se utilizarán para construir los mensajes tanto de petición como de respuesta. Estas estructuras de datos pueden construirse con cualquier lenguaje, pero lo más normal es hacerlo con XML Schema. Este apartado, es el más complicado sobre todo cuando tengamos que construir estructuras de datos muy complejas.

[4] Anexo 1: Estructura de un documento WSDL.

- **message:** Describe los mensajes que se van a intercambiar entre el cliente y el Servicio Web. Un mensaje puede estar dividido en varias partes, por ejemplo, si en un mensaje queremos enviar datos y una imagen al mismo tiempo.
- **portType:** Define el conjunto de operaciones que soporta el Servicio Web. Una operación no es más que un grupo de mensajes que serán intercambiados. Cada operación puede enviar o recibir al menos un mensaje cada vez.

En WSDL 1.1 existen 4 tipos de operaciones:

- **Unidireccional:** El Servicio recibe un mensaje y no genera ninguna respuesta.
- **Petición / Respuesta:** El Servicio recibe un mensaje y responde con otro.
- **Solicitud / Respuesta:** El Servicio envía un mensaje y recibe una respuesta.
- **Notificación:** El Servicio envía un mensaje, y no recibe respuesta.

Es importante destacar que aunque WSDL 1.1 define los 4 tipos de operaciones, sólo soporta las 2 primeras.

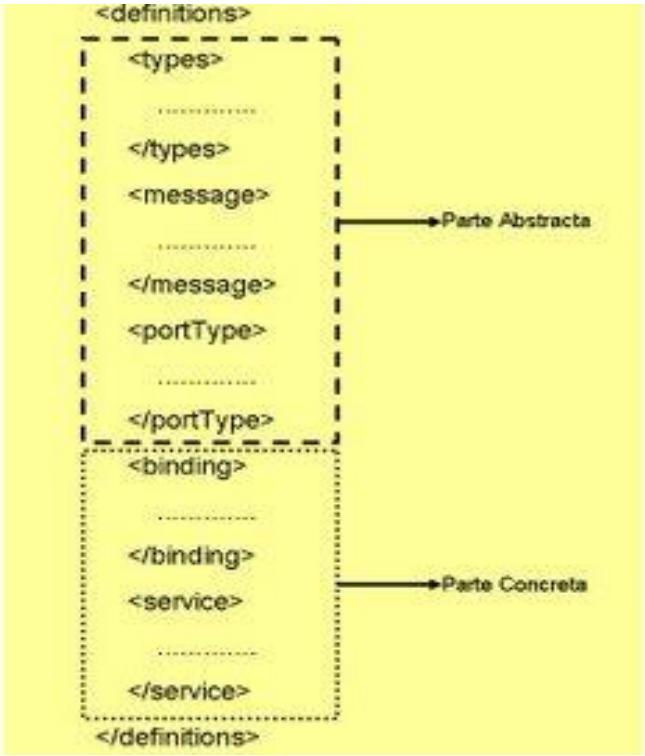


Figura 1: Esquema simplificado de un documento WSDL.

En la parte concreta tenemos:

- **binding:** Describe como formatear los mensajes para interactuar con un Servicio determinado. WSDL no define un estándar para formatear mensajes. Para ello utiliza la extensibilidad para definir como intercambiar los mensajes usando SOAP, HTTP, MIME, etc.
- **services:** Este elemento indica donde se encuentra el Servicio usando la etiqueta. Cada etiqueta define el formato de los mensajes, y la dirección donde se encuentra el servicio que acepta mensajes en ese formato.

De esta forma, la interfaz de un Servicio es el componente fundamental ya que define lo que ofrece el Servicio. Es muy importante dedicarle todo el tiempo necesario, ya que cualquier modificación provocará cambios en el resto de las aplicaciones.

Los servicios suelen estar relacionados dependiendo de su funcionalidad es por esto que se definieron los siguientes tipos de servicios para el proyecto:

- **Servicios controladores:** Serán 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. Si analizamos bien este tipo de servicios, nos daremos cuenta de que representan a los procesos de negocio que queremos 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:** Serán los servicios que representen una tarea del negocio, y que formen parte de un proceso de negocio. Este tipo de servicios serán poco reutilizables porque estarán orientados a resolver una tarea muy puntual.
- **Servicios de utilidad:** Serán aquellos servicios que se caractericen por representar una tarea altamente reutilizable. Se definirán 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 (ejemplo: servicio de acceso a bases de datos).

Una SOA está formada además por otros componentes esenciales, necesita un registro para el descubrimiento y publicación de los servicios, UDDI, un lenguaje de descripción de los servicios para su implementación, WSDL (ya fue explicado anteriormente), un protocolo de comunicación o transporte de información SOAP, un BPM para definir y proporcionar los procesos y las funcionalidades de servicios en la aplicación y un ESB que es un elemento de la integración que combina mensajería, servicios web, transformación de datos, enrutación, políticas de seguridad y que permite la interacción entre diversas aplicaciones o sistemas de una Arquitectura Empresarial.

2.4.3 *UDDI (Universal Description Discovery and Integration).*

Como resultado de la evolución de la programación orientada a objetos y el modelo de programación de componentes, se crea la necesidad de tener un lugar donde estos componentes y objetos se clasifiquen y se registren. Todo esto con el fin de que un cambio de localización no resulte en un cambio del código y se facilite el descubrimiento y el uso de los componentes. **(Tarquino, 2003)**.

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. Se puede utilizar sistemas taxonómicos estándar para clasificar estos datos y poder encontrarlos posteriormente en función de la categorización. Lo más importante es que UDDI 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.

La información en UDDI se aloja en nodos, que pueden ser privados o públicos. En la actualidad existen dos nodos públicos muy utilizados mundialmente que son los de Microsoft e IBM. Aunque una empresa pueda implementar su propio UDDI para uso exclusivo de su intranet, es importante mencionar que no existen requisitos de implementación para estos nodos, ya que se puede usar cualquier lenguaje o herramienta. Por otro lado la iniciativa UDDI consiste en ver qué datos se almacenan en UDDI y cómo se estructuran. UDDI es relativamente ligero; se ha diseñado como registro, no como depósito. La diferencia, aunque sutil, resulta esencial. Un registro redirige al usuario a recursos, mientras que un depósito sólo almacena información.

WSDL y UDDI.

WSDL se ha convertido en una pieza clave de la pila de protocolos de los servicios Web. Por eso, es importante saber cómo colaboran UDDI y WSDL y por qué la idea de interfaces frente implementaciones forma parte de cada protocolo. WSDL y UDDI se diseñaron para diferenciar claramente los metadatos abstractos y las implementaciones concretas. Para entender cómo funcionan WSDL y UDDI resulta esencial comprender las consecuencias de esta división.

WSDL distingue claramente los mensajes de los puertos: los mensajes (la sintaxis y semántica que necesita un servicio Web) son siempre abstractos, mientras que los puertos (las direcciones de red en las que se invoca al servicio Web) son siempre concretos. No es necesario que un archivo WSDL incluya información sobre el puerto. Un archivo WSDL puede contener simplemente información abstracta de interfaz, sin facilitar datos de implementación concretos, y ser válido. De este modo, los archivos WSDL se separan de las implementaciones.

UDDI establece una distinción similar entre la abstracción y la implementación con el concepto de tModels. La estructura tModel, abreviatura de "Technology Model" (modelo de tecnología), representa huellas digitales técnicas, interfaces y tipos abstractos de metadatos. El resultado de los tModels son las plantillas de enlace, que son la implementación concreta de uno o más tModels. Dentro de una plantilla de enlace se registra el punto de acceso de una

implementación particular de un tModel. Del mismo modo que el esquema de WSDL permite separar la interfaz y la implementación, UDDI ofrece un mecanismo que permite publicar por separado los tModels de las plantillas de enlace que hacen referencia a ellos.

2.4.4 *SOAP (Service Oriented Architecture Protocol).*

Es un protocolo para el intercambio de mensajes basados en XML (normalmente bajo HTTP), permite la comunicación entre servicios web, SOAP es el primer protocolo de su tipo que ha sido aceptado prácticamente por todas las grandes compañías de software del mundo. Compañías que en raras ocasiones cooperan entre sí están ofreciendo su apoyo a este protocolo. Algunas de las mayores Compañías que soportan SOAP son Microsoft, IBM, SUN, Microsystems, SAP, entre otras.

Algunas de las Ventajas de SOAP son:

- **No está asociado con ningún lenguaje:** Los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista pero los desarrolladores responsables de mantener antiguas aflicciones heredadas podrían no poder hacer esta elección sobre el lenguaje de programación que utilizan. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación.
- **No se encuentra fuertemente asociado a ningún protocolo de transporte:** La especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- **No está atado a ninguna infraestructura de objeto distribuido:** La mayoría de los sistemas de objetos distribuidos se pueden extender, y ya están alguno de ellos para que admitan SOAP.
- **Aprovecha los estándares existentes en la industria:** Los principales contribuyentes a la especificación SOAP evitaron, intencionadamente, reinventar las cosas. Optaron por extender los estándares existentes para que coincidieran con sus necesidades. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su

propio sistema de tipo que ya están definidas en la especificación esquema de XML. Y como ya se ha mencionado SOAP no define un medio de transporte de los mensajes; los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.

- **Permite la interoperabilidad entre múltiples entornos:** SOAP se desarrolló sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dicho estándares pueden comunicarse mediante mensajes SOAP con aplicaciones que se ejecuten en otras plataformas. Por ejemplo, una aplicación de escritorio que se ejecute en una PC puede comunicarse con una aplicación del back-end ejecutándose en un mainframe capaz de enviar y recibir XML sobre HTTP.

2.4.5 *ESB (Enterprise Service Bus).*

El ESB se ha convertido en una de las partes más importantes de una Arquitectura SOA ya que se presenta como el "mediador" multiprotocolo y multipropósito para dicha arquitectura.

Las aplicaciones y características más comunes del ESB son las siguientes:

- **Enrutación basada en contenido:** Generalmente, el ESB cumple un cometido de enrutación entre sistemas o aplicaciones. Esa enrutación o decisión del punto final (endpoint) que debe invocarse se puede hacer en función del contenido del mensaje. Para ello, el mensaje debe llevar información asociada que permita determinar el endpoint en cuestión (es lo que suele llamarse metadatos o metainformación del mensaje).
- **Transformación de mensajes:** La mayoría de los ESB suelen incorporar motores de transformación y parseadores de mensajes que permiten y facilitan la transformación de los mismos. La manera más común suele ser por medio de un fichero XSLT de transformación de mensajes, aunque algunas herramientas utilizan otros sistemas, incluso propietarios.
- **Configuración y no codificación:** Una de las cosas más ventajosas de un ESB es que suele realizarse todo por medio de configuración y no codificación. Independientemente del producto que estemos utilizando (hoy en día la mayoría incluyen IDE's gráficos que facilitan la configuración de los elementos del ESB; aunque algunos pueden necesitar de

configuración por medio de ficheros y manualmente) suele ser suficiente con la configuración de los componentes del ESB que vayamos a utilizar. No suele necesitarse la codificación, lo que limita el número de errores y da fiabilidad al bus que se monte. No obstante, si fuera necesario, la codificación por medio de cualquier lenguaje suele ser posible para casos concretos en que los componentes del ESB manejado no implementen o faciliten la configuración deseada.

- **Proxy de Servicios:** Una de las mayores ventajas del ESB es la de crear una capa de proxy por encima del servicio. Ello, nos permite abstraer su implementación (que al final, puede ser como una caja negra, de la que solo disponemos la interfaz, normalmente su WSDL) y añadirle información o requisitos para su ejecución.
- **Conversión de protocolos:** Como ya hemos comentado, una de las características del ESB es su funcionalidad multiprotocolo. Esto es, que permite la conexión a distintas aplicaciones o sistemas por medio de numerosos protocolos. Además, suelen incorporar la posibilidad de añadir conectores a todo tipo de aplicaciones que pueden ser desarrollados por el mismo fabricante del ESB, por terceros o incluso por el propio desarrollador a través de un API estandarizada.
- **Auditorías y Logs de Mensajes:** Otra característica importante del ESB es su función como auditor de mensajería y "logado" de mensajes. La comunicación de información a través del ESB nos permite centralizar sistemas de log y persistencia de información susceptible de ser auditada. Además, luego con herramientas de BAM o incluso Business Intelligence pueden explotarse dichos datos y realizar monitorizaciones exhaustivas que permitan la optimización del ESB, la Arquitectura SOA y en definitiva el negocio de la empresa.
- **Manejo de Excepciones:** Generalmente los servicios web tienen un comportamiento de respuesta correcta y en algunos casos pueden devolver excepciones de manera controlada (o no). El ESB nos permite gestionar las excepciones de una manera estandarizada (por ejemplo SOAP Fault) y realizar las correcciones oportunas al ocurrir dichos errores.
- **Acceso a Repositorio de Servicios**: Un Repositorio de Servicios (no confundir con UDDI) nos permite acceder a metainformación de los servicios integrados que facilita el control de los mismos y el poder aplicar políticas y condiciones de una manera centralizada y

jerárquica (entre muchas otras características, muy ligadas a veces con la gobernabilidad de la arquitectura).

- **Validación, Enriquecimiento, Transformación y Operación de Mensajes:** Se puede decir, que estas son las 4 características estrella de un ESB:

  Validación de la información, de los mensajes entrantes, sobre los que se pueden aplicar distintas condiciones y verificaciones que hagan cumplir los requisitos establecidos. **Enriquecimiento** de los mensajes, posibilidad de añadir información o metainformación adicional para cumplir las necesidades de integración en cada momento. **Transformación** de los mensajes, normalmente entre los diversos modelos de datos del conjunto de servicios y aplicaciones a integrar.
  Operación de los mensajes o enrutación en función de directivas preestablecidas o a partir de metainformación incluso en la propia comunicación.

2.4.6 *BPM (Business Process Management).*

Es el conjunto de servicios y herramientas que facilitan la administración de procesos de negocio. Por administración de procesos entendemos: análisis, definición, ejecución, monitoreo, y control de los procesos.

A través del modelado de las actividades y procesos puede lograrse un mejor entendimiento del negocio y muchas veces esto presenta la oportunidad de mejorarlos. La automatización de los procesos reduce errores, asegurando que los mismos se comporten siempre de la misma manera y dando elementos que permitan visualizar el estado de los mismos. La administración de los procesos permite asegurar que los mismos se ejecuten eficientemente, y la obtención de información que luego puede ser usada para mejorarlos. Es a través de la información que se obtiene de la ejecución diaria de los procesos, que se puede identificar posibles ineficiencias en los mismos, y actuar sobre las mismas para optimizarlos.

BPM además contempla soporte para interacción humana, e integración de aplicaciones, y es aquí la diferencia fundamental con la tecnología de WorkFlow[5] (Flujo de trabajo) existente, que es que BPM integra en los flujos a los sistemas. En la práctica un flujo BPM (o modelo de proceso BPM) visualmente es muy parecido a un WorkFlow, la diferencia está en que uno puede notar que ciertas actividades son realizadas por personas, y otras son actividades sistematizadas (realizadas por sistemas), y ambas aparecen en el flujo.

BPM también es vista como una disciplina de administración, que requiere que las organizaciones se cambien a un pensamiento centrado en los procesos y que reduzcan su dependencia de estructuras tradicionales de territorio y funcionalidad. Es un enfoque estructurado que emplea métodos, políticas, métricas, prácticas de administración, y herramientas de software para mejorar la agilidad y el desempeño operacional.

- **BPM y ESB.**

Existe un punto en que un ESB, por ser específico para la capa intermedia de una SOA podría ser más poderoso y de mejor comportamiento en algunos casos que las herramientas que hoy en día las suites de BPM incorporan. Por decirlo de otra forma, al ser soluciones creadas para un rol especifico, en general deberían dar mas soluciones a temas que están fuera del alcance SOA y WebService en términos de integración. Las soluciones ESB incluyen más que SOA y WebService, son orquestadores de servicios, permiten interactuar de mejor forma y más rápidamente a las técnicas de información con aquellas relacionadas a los procesos de negocio de las capas superiores o más altas como las de tipo presentación o BPM.

Pero. ¿Cuál seria el riesgo del uso de ESB si lo pensamos fuera del componente BPM? Principalmente se tiende a confundir las capas de middleware con las reglas de negocio, algo similar cuando los WebService compuestos se vuelven muy pesados y terminan siendo más que compuestos. Esto significa que empiezan a manejar parte de la capa de lógica de negocio, que no debería estar allí.

[5] Anexo 2: Ejemplo de WorkFlow.

BPM será el componente que deberá concentrar los procesos de negocio, las reglas de negocio, la lógica de negocio. El rol de un ESB no trata con lógica de negocio, trata con lógica de servicios, esto es resolución de servicios, monitorización de servicios, acceso de servicios, homogeneización de los mismos, Administración de adaptadores, ETLs, y todo este tipo de ámbitos. Básicamente el ESB deberá ser lo más neutral posible en cuanto a términos de negocio (lógica de negocio y procesos de negocios). Se debe considerar un medio de trasporte y coordinador de fuentes de información.

2.5 Otros elementos de SOA.

Una SOA presenta una forma de construir sistemas distribuidos que entreguen a la aplicación funcionalidad como servicios para aplicaciones de uso final u otros servicios. En la figura 2 se muestra una generalización de la arquitectura y los elementos que podrían observarse en una arquitectura orientada a servicios incluyendo los ya mencionados.

Como se puede observar, se diferencian dos zonas en la figura, una que abarca los aspectos funcionales de la arquitectura y otra que abarca aspectos de calidad de servicio. A continuación se describen los elementos brevemente:

- **Transporte:** Es el mecanismo utilizado para llevar las demandas de servicio desde un consumidor de servicio hacia un proveedor de servicio, y las respuestas desde el proveedor hacia el consumidor.
- **Protocolo de comunicación de servicios:** Es un mecanismo acordado a través del cual un proveedor de servicios y un consumidor de servicios comunican qué está siendo solicitado y qué está siendo respondido.

Funciones:

- **Descripción de servicio:** Es un esquema acordado para describir qué es el servicio, cómo debe invocarse, y qué datos requiere el servicio para invocarse con éxito.
- **Servicios:** describe un servicio actual que está disponible para utilizar.
- **Procesos de Negocio:** Es una colección de servicios, invocados en una secuencia particular con un conjunto particular de reglas, para satisfacer un requerimiento de negocio.

- **Registro de Servicios:** Es un repositorio de descripciones de servicios y datos que pueden utilizar proveedores de servicios para publicar sus servicios, así como consumidores de servicios para descubrir o hallar servicios disponibles.

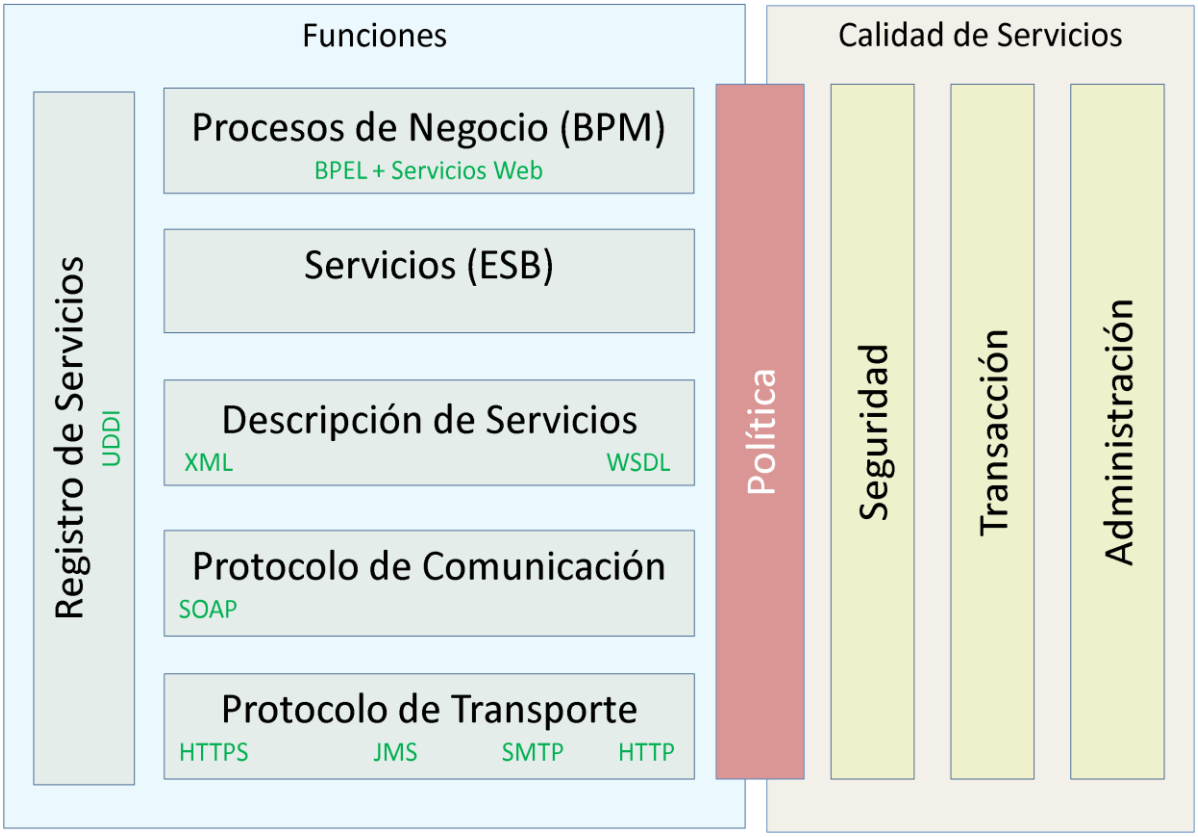


Figura 2: Componentes de SOA.

Calidad de servicio.

- **Política:** es un conjunto de condiciones o reglas bajo las cuales un proveedor de servicio hace el servicio disponible para consumidores.
- **Seguridad:** es un conjunto de reglas que pueden aplicarse para la identificación, autorización y control de acceso a consumidores de servicios.
- **Transacciones:** es el conjunto de atributos que podrían aplicarse a un grupo de servicios para entregar un resultado consistente.
- **Administración:** es el conjunto de atributos que podrían aplicarse para manejar los servicios proporcionados o consumidos.

2.6 Colaboraciones en SOA.

Las colaboraciones en SOA siguen el paradigma *find*, *bind and invoke*, donde un consumidor de servicios realiza la localización dinámica de un servicio consultando el registro de servicios para hallar uno que cumpla con un determinado criterio. Si el servicio existe, el registro proporciona al consumidor la interfaz de contrato y la dirección del servicio proveedor. **(Alvez, et al., 2006)** En la siguiente figura se muestra los roles y operaciones de una arquitectura orientada a servicios.

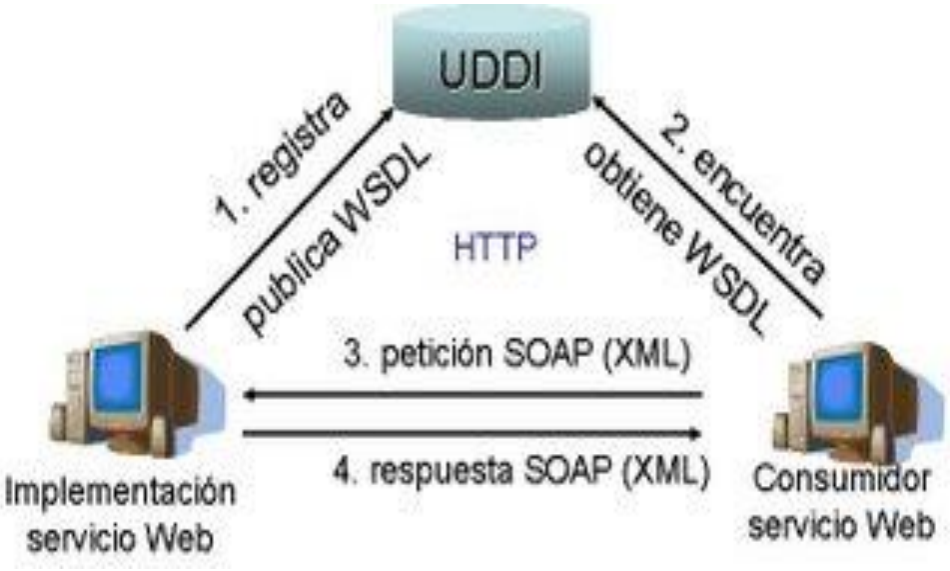


Figura 3: Colaboraciones y roles de una SOA.

Los roles son:

- **Cliente del servicio:** Es el que solicita la ejecución del servicio web, y por lo tanto el que lo consume.
- **Proveedor del servicio:** Es el encargado de implementar el servicio web y ofrecerlo a los clientes.
- **Registro del servicio:** Es un repositorio donde se almacenan las descripciones de los servicios, para que así los clientes puedan buscar el servicio web que mejor se adapte a sus necesidades.

La secuencia de ejecución más detallada es la siguiente:

1. El proveedor del servicio da de alta el servicio web en el registro. Para realizar esto, el proveedor almacena en el registro el documento de descripción de este.
2. El solicitante del servicio busca en el registro un servicio web que pueda adaptarse a sus necesidades.
3. Una vez seleccionado el servicio, el solicitante lo invoca mediante el envío de un mensaje SOAP, en el cual se indica la acción a realizar y los datos de entrada.
4. El servicio web recibe la petición y ejecuta la funcionalidad. Para finalizar envía un mensaje SOAP al solicitante con los resultados obtenidos.

2.7 Conclusiones.

De forma general se realizó un estudio de los principales componentes y herramientas de la propuesta. Se ha definido una plataforma de desarrollo completamente libre acorde con las necesidades de la universidad, así como un framework de desarrollo que ayudará a agilizar en tiempos de implementación. El capítulo 2 como tal ha sido una descripción de la propuesta que se define para el Portal de Servicios Comunitarios.

Capítulo 3

Desarrollo de la propuesta

3.1 Introducción.

Hasta ahora ha quedado claro que el principal elemento de una SOA son los servicios. Pues bien, en este capítulo se realizará una definición e identificación de los servicios más comunes necesarios para el portal. Se definirá la estructura de SOA. Se escogerá y propondrá un ESB y un BPM totalmente libres que ayuden en la gestión, comunicación y monitorización de los servicios y procesos. Por último se propondrá un modelo de la arquitectura SOA que se ha presentado durante el trabajo.

3.2 Definición e identificación de servicios.

En el capítulo 2 se hizo un estudio de las características de los servicios de una SOA. Para describirlos la mejor opción es WSDL, para su comunicación el protocolo SOAP bajo transporte HTTP y para su registro UDDI. Son los tres estándares más comunes que se usan en una SOA. Pues bien ya se tienen los estándares, entonces quedaría definir e identificar los servicios para el portal. En el capítulo 2 se hizo un estudio los tipos de servicios que existen, controladores, de negocio y de utilidad de negocio o tecnológicos (estos son los reutilizables, comunes para varias aplicaciones).

Servicios Comunitarios consta de dos grupos de módulos y seis módulos entre ambos grupos. En este trabajo solo se identificarán y definirán los servicios de tipo *utilidad* que se consideran más importantes desde el punto de vista de su reutilización para el portal, no siendo así el caso de los de *negocio* y los *controladores* debido a que estos son mas específicos en su funcionalidad y serían un gran número a la hora de definirlos.

Por otra parte en la universidad la digitalización juega un papel importante, la gran mayoría de las acciones cotidianas que realizan las personas vinculadas al centro son a través de un gran número de portales o aplicaciones web que se comunican para brindar los servicios

necesarios que se llevan a cabo en la vida cotidiana de la UCI. Para ello todo el personal que requiera tener acceso a este tipo de aplicaciones cuenta con un “usuario UCI” necesario para interactuar con la aplicación. De aquí que los principales servicios que se definirán serán los vinculados a la gestión del “usuario UCI” en las aplicaciones web del portal.

Servicios de utilidad:

Protocolo: **SOAP.**
Estilo por defecto: **rpc.**
Trasporte: **SOAP sobre HTTP.**

Métodos:

Autenticar Usuario.

Descripción: Autenticar un usuario dado ID y Password.

Tipo de Operación: Unidireccional (El Servicio recibe un mensaje y no genera ninguna respuesta)

Input:

IdUsuario tipo String, Password tipo String

Métodos:

Buscar Usuario.

Descripción: Realizar una búsqueda de los datos de uno o varios usuarios en especifico.

Tipo de Operación: Petición / Respuesta (El servicio recibe un mensaje, y envía la respuesta)

Input:

IdUsuario tipo String, Cadena tipo String

Output:

BuscarUsuarioReturn tipo UsuarioArray - arreglo de tipo Usuario
NombreUsuario tipo String
IdUsuario tipo Integer
ClUsuario tipo Integer
FacultadUsuario tipo Integer
GrupoUsuario tipo Integer
ProvinciaUsuario tipo String
MunicipioUsuario tipo String
SolapinUsuario tipo Integer
FotoUsuario tipo String
TipoUsuario tipo String

Métodos:

Insertar Usuario.

Descripción: Insertar un usuario con un rol correspondiente.

Tipo de Operación: Unidireccional (El Servicio recibe un mensaje y no genera ninguna respuesta)

Input:

IdUsuario tipo String, Rol tipo String

Métodos:

Eliminar Usuario.

Descripción: Elimina un usuario dado su Id.

Tipo de Operación: Unidireccional (El Servicio recibe un mensaje y no genera ninguna respuesta)

Input:

IdUsuario tipo Integer

Métodos:

Modificar Usuario.

Descripción: Modifica un usuario dado su Id.

Tipo de Operación: Unidireccional (El Servicio recibe un mensaje y no genera ninguna respuesta)

Input:

IdUsuario tipo Integer, Rol tipo String

Métodos:

Registrar acciones de un usuario.

Descripción: Devuelve todas las acciones de los usuarios en la aplicación dado una fecha.

Tipo de Operación: Petición / Respuesta (El Servicio recibe un mensaje y responde con otro)

Input:

Fecha_Hora tipo DataTime

Output:

BuscarRegistroReturn tipo RegistroArray - arreglo de tipo Registro
NombreUsuario tipo String
IdUsuario tipo String
Ip tipo String
Motivo tipo String
HistorialArray (arreglo) **tipo Historial**

3.3 Organización estructural de SOA.

Para el desarrollo de una SOA primeramente se debe definir un ciclo de vida que facilite la simplificación y organización del trabajo. En el capítulo 2 se hizo un estudio del ciclo de vida para una SOA que por lo general es el más eficiente y respaldado por grandes empresas como IBM, la cual usa uno similar. Durante las fases definidas en el ciclo de vida tomará lugar la construcción de la SOA.

SOA propone un modelo de pensamiento con el cual atacar la definición de arquitectura, y no las de aplicaciones individuales, sino las arquitecturas de organizaciones completas. Desde el punto de vista tecnológico, SOA propone varias capas de servicios que exponen

funcionalidad, fundamentalmente de negocio, que permiten la composición de aplicaciones a partir de los mismos. La SOA estará constituida por varias capas:

1. **Capa de Presentación:** Será la encargada de interactuar con los usuarios de la aplicación, expondrá los datos necesarios y se encargará de la manipulación de entrada y recepción de datos.
2. **Capa de Servicios:** Una característica de esta capa es que puede ser movida para atender requerimientos de tiempos y rendimientos sin afectar el funcionamiento de la aplicación, debido a la independencia que tiene con el usuario. Por lo general esta capa puede ser subsidiada o subdividida en varias capas entre el cliente y el servidor. Alberga lógica de negocio, seguridad, metadatos, etc.
3. **Capa de Acceso a Datos:** Se encargará de todo el procesamiento de datos que necesiten las capas superiores. Por lo general a través de un ORM.

Como toda arquitectura que siga el patrón de capas, cada capa tendrá su nivel de responsabilidades. Las capas superiores solicitarán servicios a las capas inmediatamente inferiores. Este paradigma de estructura por capas proporciona las ventajas de reutilización, flexibilidad, mantenimiento y escalabilidad. Permite además dividir proyectos complejos y largos en varios más pequeños y a su vez más fáciles y seguros de elaborar.

3.3.1 *Capa de Presentación. Componentes.*

Es la que ve el usuario, presenta el sistema al usuario, le comunica la información y captura la información del usuario en 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. También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Para la implementación de esta capa se propone utilizar un Sistema de Gestión de Contenidos (CMS por sus siglas en ingles).

CMS

Es un programa que permite crear una estructura de soporte (framework) para la creación y administración de contenidos por parte de los participantes principalmente en páginas web.

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 el público.

Un sistema de administración de contenido a menudo funciona en el servidor del sitio web. Muchos sistemas proporcionan diferentes niveles de acceso dependiendo del usuario, variando si es el administrador, editor, o creador de contenido. El acceso al CMS es generalmente vía el navegador, y a veces se requiere el uso de FTP para subir contenido, generalmente fotografías o audio.

Los creadores de contenido crean sus documentos en el sistema. Los editores comentan, aceptan o rechazan los documentos. El editor en jefe es responsable por publicar el trabajo en el sitio. El CMS controla y ayuda a manejar cada paso de este proceso, incluyendo las labores técnicas de publicar los documentos a uno o más sitios. En muchos sitios con CMS una sola persona hace el papel de creador y editor, los blogs generalmente funcionan de esta manera.

Algunos de los CMS open source mas utilizados son Zone/Plone, Drupal, Typo3, entre otros. Cualquiera de estos pueden ser utilizados lo importante es que se adapte a las necesidades de los desarrolladores, el sistema a desarrollar y que permita la creación de componentes dinámicos.

Patrones

Se propone para esta capa el uso del patrón Modelo Vista Controlador (MVC por sus siglas en ingles), con el objetivo de una mejor organización de las clases. Una de sus ventajas es que permite separar responsabilidades entre las clases, lo que ayudará a la escalabilidad de la interfaz y la reutilización de componentes.

Modelo: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo,

no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o importes en un carrito de la compra.

Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos. En MVC corresponde al modelo realizar esa tarea. Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el. Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo. El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se

actualice. Sin embargo, en algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

3.3.2 *Capa de Servicios. Componentes.*

La capa de servicios de una arquitectura SOA tiene la responsabilidad de modelar la lógica de negocios, ofrecer funcionalidad de la aplicación, minimizar dependencias entre el negocio y la aplicación a través de los servicios y reutilizar estos servicios de negocio en un Bus de Servicios (ESB por sus siglas en ingles). Esta capa además proporcionara servicios a la Capa de Presentación en dependencia de las solicitudes del usuario.

Los servicios creados modelan la empresa según múltiples niveles de abstracción, y por tanto SOA se dice que tiene varios niveles de servicios: **(The Server Labs)**

Servicios de aplicación: Los servicios de aplicación son los servicios de más bajo nivel y nos sirven como base para crear servicios de negocio. Los servicios de aplicación proporcionan servicios de utilidad independientes de la solución.

Servicios de Negocio: Los servicios de negocio son los responsables de expresar la lógica de negocio. Pueden encapsular en un servicio la lógica de una tarea o proceso de negocio específico o representar una entidad de negocio específica. Estos servicios son, por su gran valor añadido, directamente publicables en el ESB.

Servicios de Orquestación: Los servicios de orquestación consisten en servicios de proceso compuestos por servicios de negocio y servicios de aplicación coordinados mediante reglas de negocio y secuencias de ejecución. Estos servicios son también publicables en el ESB.

Según la definición de SOA la lógica de la aplicación debe residir en servicios web accesibles a todas las aplicaciones consumidoras, la forma mas común es mantener un registro de servicios mediante UDDI. Aunque la mayoría de las empresas lo utilizan con un ESB para la integración. En este punto se estará hablando de la estructura UDDI que será el principal

registrador de servicios. También se propone el uso de un ESB open source con vista a futuras integraciones del Portal de Servicios Comunitarios.

Registro para Servicios Web. UDDI

En el capítulo 2 se estudió UDDI como el registro por excelencia de SOA. En este punto se realizará un estudio en cuanto a estructurar UDDI, ver como se registran servicios en él y como funciona en tiempo de ejecución.

Se espera que la cantidad de Servicios Web disponibles aumente de manera considerable en poco tiempo. Para administrar estos Servicios Web es necesario un registro, el cual es un lugar central donde se pueden ubicar los Servicios Web de manera categorizada. Surge la iniciativa UDDI que es el estándar para el registro de los servicios web y su propósito es facilitar el descubrimiento de servicios tanto dinámicamente como en tiempo de diseño.

Estructura de datos de UDDI.

El schema XML para UDDI define cuatro tipos de datos básicos para la información del negocio y del servicio. Todos los datos en repositorio UDDI debe pertenecer a uno de los 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
- BusinessService
- BindingTemplate
- TModel

La información de estos tipos de datos puede clasificarse en páginas blancas, amarillas y verdes de acuerdo a la Figura 4.

El operador del repositorio UDDI genera un identificador único para cada tipo de datos cuando la información es publicada por primera vez en el repositorio. La figura 4 muestra la interrelación entre los cuatro tipos de datos y como se clasifican en las páginas blancas,

amarillas o verdes. Solo se necesita un elemento del tipo BusinessEntity para publicar un servicio en un repositorio UDDI, los otros elementos son opcionales. Los cuatro tipos de datos pueden ser clasificados de manera general en *Información del negocio* e *Información del Servicio*. UDDI define una estructura compleja que contiene la información necesaria para descubrir los servicios e interactuar con ellos. La figura 5 muestra un registro UDDI como una caja que contiene una jerarquía de pequeñas cajas. Un registro UDDI contiene millones de estos registros.

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 entidad. Un elemento businessEntity contiene los siguientes atributos:

- Name: Almacena el nombre de la entidad (es obligatorio).
- Contatcs: Describe vías de contacto con la entidad (teléfonos, e-mails, Fax).
- categoryBag: Almacena los datos categorizados en el tipo de industria, tipo de producto, servicio y geografía.
- IndentifierBag: Almacena los códigos identificadores de la industria.

Otro de los campos en un businessEntity es su businessServicesList (lista de servicios del negocio) que es una lista de estructuras del tipo businessService, donde cada una representa la funcionalidad de un Servicio Web en particular que expone la entidad.

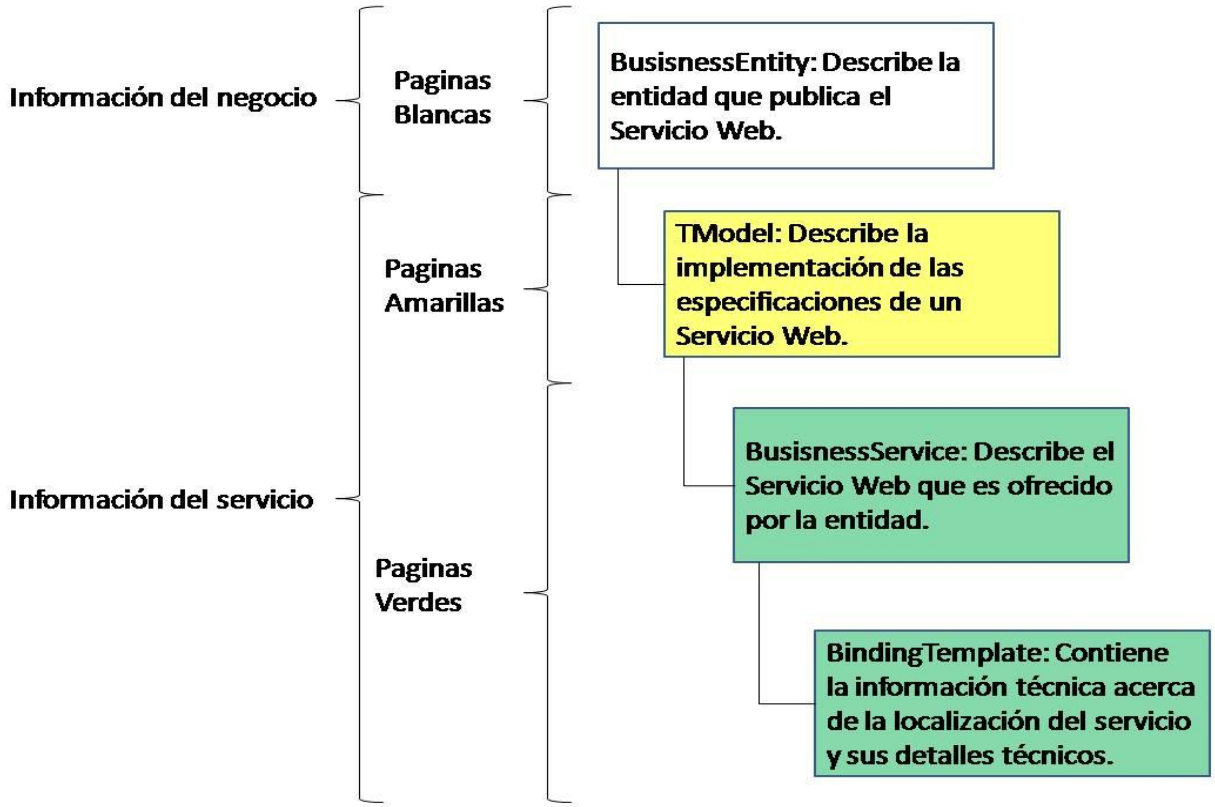


Figura 4: Tipos de datos UDDI.

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 (paginas 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 mas (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. En

la Figura 5, el punto de acceso (Access Point) provee la primera parte de la información, la cual apunta directamente al URL. Cada bindingTemplate contiene una lista de estructuras tModelInstanceInfo (esta lista es llamada tModelInstanceDetails).

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. TModels son especificaciones técnicas que describen la naturaleza de un servicio, lo más adecuado es utilizar WSDL para describir estas especificaciones. UDDI ha adoptado un marco de trabajo extensible para describir la naturaleza de un negocio. 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 Web.

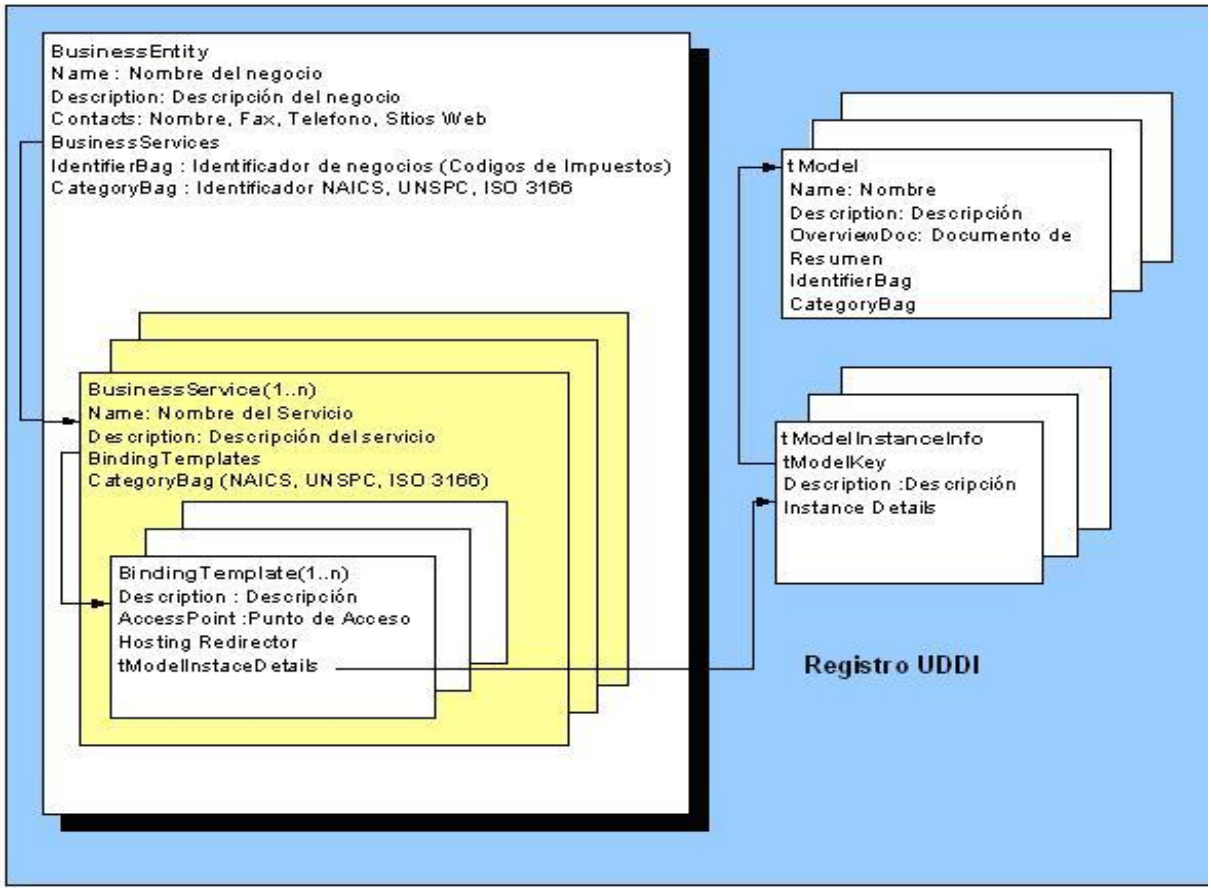


Figura 5: Detalle de los datos de UDDI.

Interfaz de programación (API) de UDDI.

¿Como funciona la base de datos UDDI? La interfaz de programación de UDDI consta de dos partes. Una API de consulta para buscar o descubrir[6] y una API para publicar o registrar[7]. 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 los APIs de publicación y consulta en la forma que la especificación sugiere que lo haga UDDI.

En UDDI existen métodos necesarios para la búsqueda en el registro:

Encontrar el negocio de interés: Es la manera lógica inicial de la mayoría de las operaciones de búsqueda. Un navegante de Internet puede dar criterios de búsqueda (como productos, categorías de productos, palabras claves, etc.) para empezar la búsqueda. La intención de incluir este método es para hacer la búsqueda preliminar, después de esto se hacen búsquedas más precisas en los pasos posteriores. Este método retorna una lista resumida de resultados. El nombre técnico para cada resumen es una estructura del tipo bussinesInfo, que es una forma abreviada de la estructura bussinessEntity.

El siguiente paso lógico es ***encontrar las categorías de los servicios que un negocio en particular ofrece***. Para esto, UDDI ofrece un método para buscar una lista abreviada de servicios que un negocio en particular ofrece. El buscador inteligente debe usar este método para buscar la lista de todos los servicios, una vez el usuario ha seleccionado el negocio de su interés.

Buscar todos los tModel de interés: El usuario debe proveer un criterio de búsqueda (es decir, una categorización del negocio, que es un criterio de búsqueda muy importante), el

---

[6] Anexo 3: Ejemplo de búsqueda en UDDI.

[7] Anexo 4: Ejemplo de registro en UDDI.

método retorna una lista de los tModels que cumplen este criterio. Los usuarios pueden buscar por fingerprints (huellas digitales) específicos usando este método. Mientras UDDI ofrece un mecanismo para descubrir los Servicios Web al mismo tiempo tiene un framework rico y extensible para la categorización de un negocio. La especificación UDDI permite el desarrollo de servicios especializados de búsqueda para entidades específicas.

Encontrar los enlaces (bindings) de interés en un servicio particular de un negocio. Cuando se invoca este método el registro UDDI responde enviando una lista de todos los binding templates que el servicio del negocio contiene. Los bindingTemplate ofrecen el access point del Servicio Web. Una binding template referencia generalmente una interfaz WSDL.

Un negocio puede implementar múltiples bindingTemplates.

Obtener el detalle de un binding template en particular. Normalmente los usuarios invocan métodos get para recuperar detalles de los resultados abreviados de los métodos find. También existen métodos para recuperar detalles de todas las estructuras antes mencionadas.

Cualquier búsqueda en un registro UDDI es una combinación de estos métodos básicos de búsqueda. Se puede pensar en conversaciones de búsqueda utilizando diferentes secuencias de los métodos básicos de búsqueda.

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

1. Usar el registro UDDI para localizar el elemento businessEntity del negocio apropiado
2. Localizar el elemento businessService para identificar los Servicios Web ofrecidos por la empresa.
3. Seleccionar el bindingTemplate para recuperar la dirección del Servicio Web y el elemento tModel para asegurar la compatibilidad técnica entre los sistemas.

ESB propuesto.

Por lo general la mayoría de los ESB que se usan en las empresas son implementados por ellas mismas para adaptarlo a sus prestaciones y necesidades. Este proceso implica mucho tiempo y esfuerzo, generalmente para cuando se ha implementado el ESB, ya las características del negocio pueden haber desviado sus requerimientos originales. El objetivo no es implementar un ESB desde cero sino utilizar uno que sea de código abierto (open source) que se adapte a las necesidades actuales. Esto permitirá en un futuro cambiar incluso el lenguaje del servidor en que fue desarrollado para que se adapte mejor a nuestras condiciones. De esta forma se propone Mule ESB.

¿Qué es Mule?

Mule es una plataforma de mensajería construida en base a ideas de la arquitectura de los Bus de Servicios Empresariales. Un ESB trabaja actuando como una especie de sistema de tránsito que acarrea data dentro o fuera de su Intranet. El ESB define una serie de paradas -endpoints- a través de las cuales, aplicaciones pueden enviar o recibir data desde o hacia el sistema. El corazón del sistema, el Bus de mensajería, se encarga de enrutar mensajes entre los endpoints **(Cooperativa DESA, 2008)**. Por otra parte Mule es:

- **Una plataforma de mensajería.** Está diseñado para la implementación de arquitecturas orientadas a servicios. El núcleo de Mule es un contenedor de servicios basado en SEDA (staged event-driven architecture). Mule administra componentes llamados UMOs (Universal Message Objects), que en la práctica son plain old java objects, lo que se traduce en una gran facilidad para la construcción de nuevos conectores a plataformas no incluidas inicialmente.
- **Un contenedor de aplicaciones.** Mule actúa como un contenedor de aplicaciones sencillo y robusto, integrado de forma transparente a sistemas externos con los estándares de comunicación más conocidos.
- **Una Plataforma para proceso masivo de mensajes.** Mule provee la plataforma ideal para la construcción de aplicaciones de procesamiento masivo de mensajes, agregando

inmediatamente a la solución la administración, monitoreo, log y configuración centralizadas, que son propias del producto.

- **Administración gráfica centralizada de los componentes[8].** La operación de cada componente por separado puede ser controlada, por ejemplo, pausando y encolando los requests para solucionar problemas eventuales en otras etapas del proceso.
- **Monitoreo de los componentes[9].** Estadísticas en tiempo real del comportamiento de los componentes pueden ser observadas en la consola del sistema.

¿Por qué Mule?

Las opciones que el proyecto Mule brinda son:

- Un framework de mensajes escalable que debe manejar las complejidades de la integración de sistemas.
- Disponer de un servidor fácil de usar, aunque poderoso, que pueda operar sobre topologías complejas.
- Desarrollo y puesta en producción de componentes de manera simple y autónoma.
- Reutilización de código. Si todos los componentes son unidades independientes y auto-contenidas, pueden ser conectados en cualquier otro sistema.
- Rápido tiempo a producción (Rapid time to market). Utilizando Mule se obtienen funcionalidades que permiten ahorrar tiempo por medio de no tener que desarrollar en algunos casos o de un bajo overhead de mantenimiento.
- Flexibilidad. Una poderosa configuración que debe ser fácil de administrar sobre ambientes distribuidos.

---

[8] Anexo 5: Vista gráfica de la administración de componentes de Mule.

[9] Anexo 6: Vista del monitoreo de componentes de Mule.

BPM propuesto.

"*Intalio* | *BPMS* es el futuro de Gestión de Procesos de Negocio. Es una herramienta de procesos web graficados basados en WorkFlows, generación automáticas de formas y de clase empresarial para arquitecturas Open Source…" **(Stadil, 2007)**.

Intalio BPM ofrece varias ventajas

- Ofrece un modelo ágil para el despliegue de procesos de negocio.
- Capacidad para unir los conceptos de Web Services y Workflow en un proceso de negocio.
- El soporte a estándares de la industria de BPM.
- Una interface de usuario gráfica muy clara que permite la fácil familiarización con la herramienta.

3.3.3 *Capa de acceso a datos.*

Esta capa tiene la función de realizar los accesos hacia la base de datos para la persistencia de la información así como también para las consultas de la misma. Generalmente los datos se guardan en un sistema gestor de bases de datos relacional. Para la persistencia se usan componentes de acceso a datos. En una SOA esta funcionalidad se puede gestionar a través de los servicios o de una forma más correcta mediante software ORM (Object Relational Mapping) como motores o frameworks de persistencia.

La principal ventaja que aporta el ORM es la reutilización, permitiendo llamar a los métodos de un objeto de datos desde varias partes de la aplicación e incluso desde diferentes aplicaciones. La capa ORM también encapsula la lógica de los datos. Un buen ORM permite:

- Mapear clases a tablas: propiedad a columna, clase a tabla.
- Persistir objetos.
- Recuperar objetos persistidos.
- Recuperar una lista de objetos a partir de un lenguaje de consulta especial.

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

ORM Propuesto. PROPEL.

PROPEL es un ORM con un marco para PHP 5. Permite acceder a una base de datos usando un conjunto de objetos, ofrece una API sencilla para almacenar y recuperar datos. **(Trac Powered, 2007)**.

Propel es un servicio de objeto persistente y de consulta, lo que significa que Propel provee un sistema para almacenar objetos en una base de datos y un sistema para búsqueda y restauración de objetos desde una base de datos. Propel permite realizar consultas complejas y manipulación de bases de datos sin escribir una sola cláusula SQL. Propel hace más fácil la escritura de aplicaciones, más fácil de desplegar, y mucho más fácil para migrar si alguna vez la situación lo amerita.

3.4 Modelo propuesto.

La colaboración que se establece mediante este modelo que se propone parte desde una petición del usuario a partir de las interfaces gráficas que representan algunos sitios de la universidad. Una vez que el usuario envía la petición en la capa de servicios donde se instancia un proceso de negocio a partir del BPM. Dicho proceso necesita de servicios que contengan la implementación para la ejecución de la petición, para ello invoca un servicio que se encuentra en un Servidor Web que se ajuste a sus necesidades. Como el ó los servicio(s) invocado(s) tienen un formato de mensajes específicos, necesita de un ESB para adecuar el formato de forma que sea entendible por los demás componentes y en específico por el proceso de negocio en curso.

El ESB además, se coordinará con el registro de servicios (UDDI) para obtener la dirección de localización (endpoint) del servicio web de forma dinámica, esto proporciona una flexibilidad a la aplicación muy importante frente a cambios de localización del los servicios.

El modelo SOA que se muestra en la figura 6 cuenta con los principales componentes que se han mencionado en el trascurso de este trabajo. De todos se hizo un estudio detallado. Además de lo componentes ya mencionados cuenta con dos adicionales, el Data Services (Servidor de Información) y el Transaction Services.

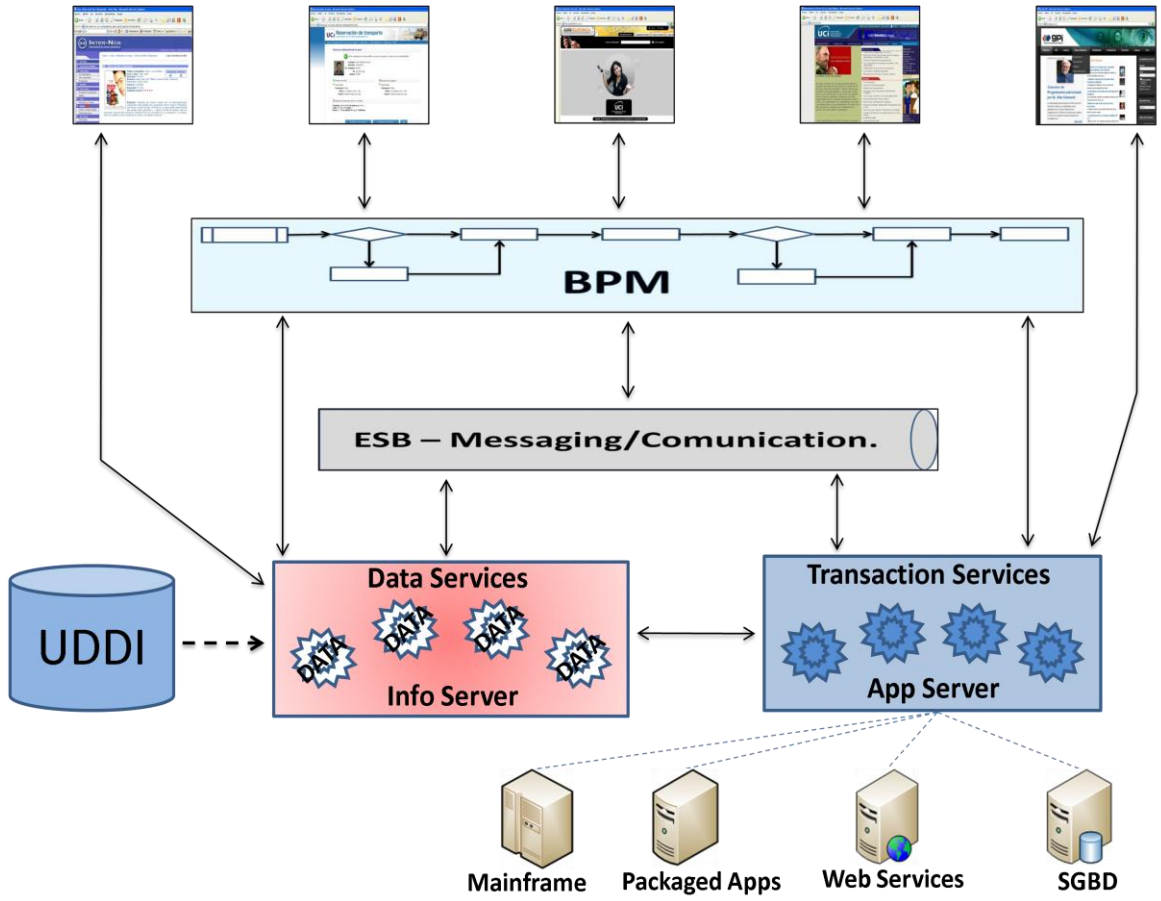


Figura 6: Modelo SOA.

Un Servidor de Información está compuesto por una capa de servicios de datos que proporciona medición o captación de datos entre los diferentes consumidores y fuentes de una empresa o proyecto. Su principal función es la simplificación del acceso a datos y las actualizaciones que requiere, permitiendo que una vez creados sean altamente reutilizables.

La transacción de servicio se aplica a un grupo deservicios determinados a través de atributos con el objetivo de devolver un conjunto de datos consistentes para establecer la comunicación

entre diversas aplicaciones o sistemas. Por medio de esta funcionalidad un entorno SOA puede utilizar servicios de sistemas mainframes reduciendo el coste y la duración de los proyectos que necesitan obligatoriamente de este tipo de sistemas.

3.5 Conclusiones.

Toda arquitectura está compuesta por herramientas, lenguajes y componentes necesarios para poder desempeñar un sistema determinado. La arquitectura SOA que se propone estará estructurada por toda una serie de servicios, capas y componentes que se han definido para su desarrollo. Se concluye con un modelo estructurado y organizado que da solución a toda la investigación que se ha realizado en este documento.

Conclusiones

Este trabajo está propuesto para el desarrollo del Portal de Servicios Comunitarios. Se ha elaborado esta propuesta fundamentándose principalmente en la necesidad que actualmente presenta la universidad y el país en general en cuanto a la migración de la mayoría de sus proyectos a Software Libre. El modelo presenta una organización estructural y sencilla que va a facilitar el desarrollo de aplicaciones web en el portal de una manera eficaz y rentable en cuanto a tiempo de optimización e implementación.

Con este trabajo se han logrado dar los primeros pasos a la futura puesta en práctica de una Arquitectura Orientada a Servicios para el Portal de Servicios Comunitarios. A partir de la presentación de un modelo y los componentes necesarios para su desarrollo.

Recomendaciones

Se recomienda profundizar más en el estudio de la propuesta de arquitectura que se ha definido para adoptar en el portal del proyecto, analizando y documentando los resultados obtenidos, con el objetivo de mejorar esta alternativa. Adoptar el ciclo de vida propuesto con el objetivo de realizar una arquitectura organizada desde su principio, analizando las particularidades y requerimientos del proyecto. Se exhorta a estudiar profundamente la arquitectura actual para reutilizar lo que se pueda, evitando tener que perder tiempo en nuevos desarrollos. Elaborar aún más el catálogo de desarrollo UDDI que actualmente comenzó a ponerse en práctica en la universidad con muy buenos resultados. Por último se propone enfatizar en la necesidad de impartir cursos de SOA y de los sistemas operativos Linux propuesto (Debian y Ubuntu), con el objetivo de aumentar el conocimiento necesario para el trabajo del personal involucrado en el proyecto.

Bibliografía citada

(Alvez, et al., 2006): Alvez, Pablo, Foti, Patricia y Scalone, Marco. 2006. ***Generador de Aplicaciones Orquestadoras.*** **Uruguay : s.n., 2006.**

(Bylli Reynoso, 2006): Bylli Reynoso, Carlos. 2006. **[En línea] Wylli.net, 26 de 6 de 2006. [Citado el: 28 de 12 de 2007.] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx.**

(Casanovas, 2004): Casanovas, Josep. 2004. **Desarroloweb.com. [En línea] 31 de 7 de 2004. [Citado el: 28 de 12 de 2007.] http://www.desarrolloweb.com/articulos/1622.php.**

(Coletti, 2006): Coletti, Daniel. 2006. **Soa y el Software libre. [En línea] 17 de 11 de 2006. [Citado el: 24 de 1 de 2008.] http://www.danielcoletti.com.ar/index.php/2006/11/17/soa-y-el-software-libre/.**

(Cooperativa DESA, 2008): Cooperativa DESA. 2008. **DESA. [En línea] 2008. [Citado el: 12 de 5 de 2008.] http://www.desa.com.ve/.**

(Debian, 2008): Debian. 2008. **Debian el sistema operativo universal. [En línea] 20 de 2 de 2008. [Citado el: 12 de 5 de 2008.] http://www.debian.org/index.es.html.**

(Garzás): Garzás, Javier. **LA ARQUITECTURA SOFTWARE. EL MODELO 4+1. [En línea] Copyright Javier Garzás. [Citado el: 17 de 2 de 2008.] http://jgarzas.googlepages.com/4mas1.**

(Molineda 2005): Mollineda, Ramón. 2005. **Arquitectura de Software: arte y oficio. [En línea] 4 de 2005. [Citado el: 11 de 2 de 2008.] http://web.iti.upv.es/actualidadtic/2005/02/2005-02-arquitectura.pdf.**

(Orlich, 1998): Orlich, Daniel. 1998. **SISTEMAS DE INFORMACIÓN GEOGRÁFICA. [En línea] SINADES, 1998. [Citado el: 29 de 12 de 2007.] http://www.mideplan.go.cr/sinades/Proyecto_SINADES/sostenibilidad/armonizacion/index-10.html#C.%20Arquitecturas%20monolíticas. ATN/SF/4717-CR.**

(Simoes): Simoes, Dorival. **El paso que sigue. [En línea] ACENTURE y HP Invent. [Citado el: 11 de 12 de 2007.] http://64.116.224.233/detras_el_paso_que_sigue/fichas/soa.html.**

(SOA Introduction, 2006): ***SOA Introduction.*** Mangarelli, Eduardo y Cabrera, Martín. 2006. **Uruguay : s.n., 2006.**

(Stadil, 2007): Stadil, Sebastean. 2007. **[En línea] 18 de 10 de 2007. [Citado el: 10 de 5 de 2008.] http://www.intalio.com.**

(Tarquino, 2003): Tarquino, Jaime. 2003. **Registro para servicios Web, UDDI y SOA . [En línea] 26 de 7 de 2003. [Citado el: 5 de 3 de 2008.] http://www.willydev.net/descargas/Articulos/JaimeTarquino/wssoa.aspx .**

(The Server Labs): The Server Labs. **¿Pero, qué es realmente SOA? [En línea] [Citado el: 4 de 26 de 2008.] http://www.theserverlabs.com/folletos/Folleto%20SOA.pdf.**

(TIBCO): TIBCO. **El papel de un bus de servicios empresariales (ESB) en una SOA. [En línea] [Citado el: 4 de 3 de 2008.] http://www.tibco.com/international/spain/resources/es_esb_for_soa.pdf.**

(Trac Powered, 2007): Trac Powered. 2007. **PROPEL. [En línea] 12 de 3 de 2007. [Citado el: 3 de 5 de 2008.] http://propel.phpdb.org/trac/.**

(Visual Paradigm International, 2007): Visual Paradigm International. 2007. **Free Download Manager. [En línea] 5 de 3 de 2007. [Citado el: 12 de 4 de 2008.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p/.**

Bibliografía consultada

Collado, Cecilia. 9. **SOA: Primeros pasos.** ***The GBM Journal.*** **34, 9, Vol. III.**

Daccach T., José Camilo. 2006. **Articulos Delta. [En línea] J.C.Daccach T. 1997 - 2008., 3 de 1 de 2006. [Citado el: 4 de 3 de 2008.] http://www.deltaasesores.com/prof/PRO152.html.**

Desarrollador Web. **desarrolladorWeb.com. [En línea] [Citado el: 16 de 3 de 2008.] http://www.desarrolloweb.com.**

Gómez, Alvaro. 2008. **ESB... El corazón de SOA. [En línea] 27 de 1 de 2008. [Citado el: 12 de 5 de 2008.] http://alvarogomezrubio.blogspot.com/2008_01_01_archive.html.**

Grehan, Rick. 2006. **InfoWorld. [En línea] 17 de 10 de 2006. [Citado el: 24 de 2 de 2008.] http://www.iworld.com.mx/iw_TestCenter_read.asp?iwid=112.**

Internea. 2005. **Programas de capacitacion/GNU-Linux. [En línea] 23 de 4 de 2005. [Citado el: 5 de 3 de 2008.] http://linux.internea.com.ar/mod/resource/view.php?id=91.**

Microsoft. **Microsoft.** ***Centro para la mediana empresa.*** **[En línea] [Citado el: 24 de 2 de 2008.] http://www.microsoft.com/mexico/empresas/softnews/marzo/soluciones2.mspx.**

—. 2006. **Microsoft Corporation. [En línea] 12 de 2006. [Citado el: 11 de 2 de 2008.] : http://download.microsoft.com/download/c/2/c/c2ce8a3a-b4df-4a12-ba18-7e050aef3364/070717-Real_World_SOA.pdf.**

Paper, Whiter. 2006. **[En línea] 2 de 2006. [Citado el: 4 de 3 de 2008.] http://blogs.sun.com/jaimecid/resource/BPM-Spain_Sun_SOA_20061010.pdf.**

Pedro. 2008. **Espacio SOA.net. [En línea] 10 de 1 de 2008. [Citado el: 14 de 3 de 2008.] http://www.espacioSOA.net.**

Reynoso, Carlos y Kicillof, Nicolás. 2006. **Lenguajes de Descripción de Arquitectura (ADL). [En línea] Microsoft, 26 de 6 de 2006. [Citado el: 18 de 2 de 2008.] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/lenguaje.mspx#E2G.**

SOA agenda. **SOA agenda. [En línea] [Citado el: 28 de 4 de 2008.] http://www.soaagenda.com.**

Software AG. 2008. **[En línea] copyright © 2008 software ag, 2008. http://www.softwareag.com.**

—. **Registros y repositorio SOA basados en estándares abiertos en el corazón mismo de la suite crossvision. [En línea] [Citado el: 2 de 3 de 2008.] http://www.softwareag.com/es/Images/FS_Centrasite_ESP_Q206_tcm24-18029.pdf.**

2003. **Tienda Linux.com. [En línea] 31 de 5 de 2003. [Citado el: 27 de 2 de 2008.] http://soporte.tiendalinux.com/portal/Portfolio/postgresql_ventajas_html.**

Ubuntu. **Linux para los seres humanos. [En línea] [Citado el: 12 de 5 de 2008.] http://ubuntu.com.es/.**

Wikimedia Foundation. 2008. **Wikipedia. La Enciclopedia Libre. [En línea] 12 de 1 de 2008. [Citado el: 29 de 12 de 2007.] La fecha es acorde a la última actualización del sitio. http://es.wikipedia.org/wiki/Arquitectura_de_software.**

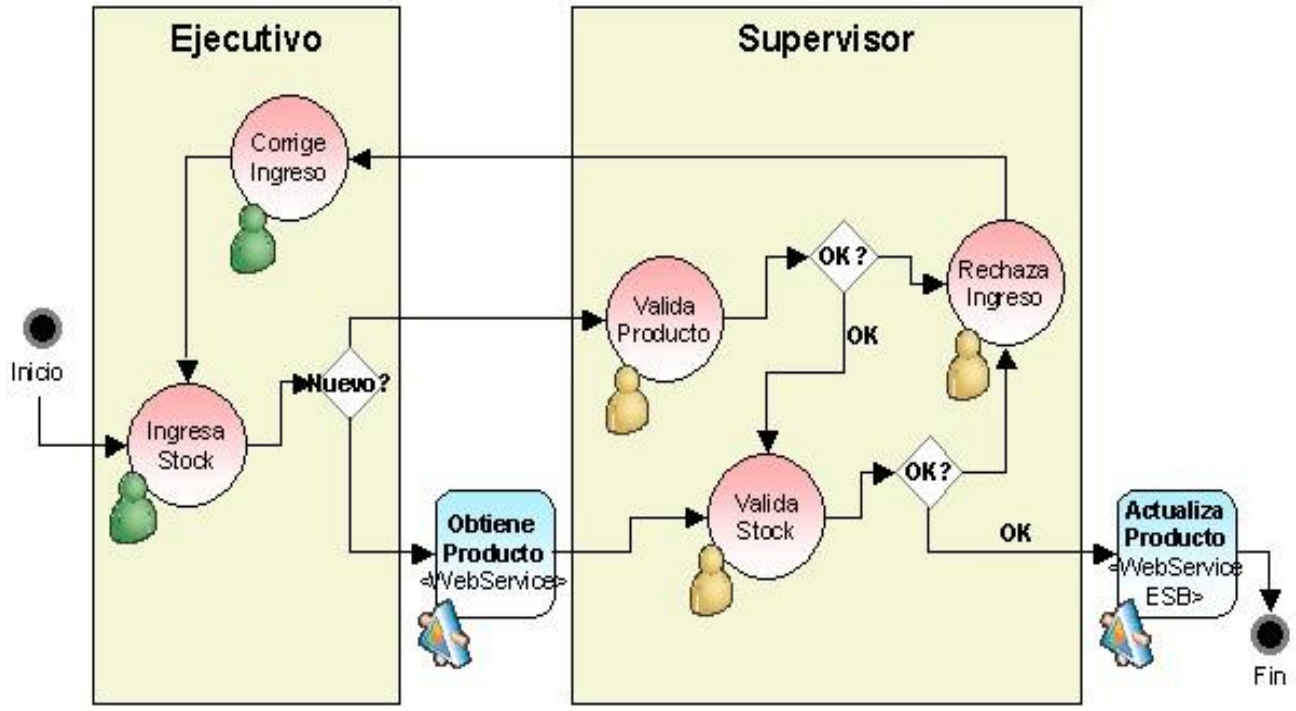
Anexos

Anexo 1: Estructura de un documento WSDL.

```
<?xml version ='1.0' encoding ='UTF-8' ?>
<definitions name='getData'
     targetNamespace='getData'
     xmlns:tns='getData'
     xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
     xmlns:xsd='http://www.w3.org/2001/XMLSchema'
     xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
     xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
     xmlns='http://schemas.xmlsoap.org/wsdl/'>
<message name='getDataRequest'>
    <part name='symbol' type='xsd:string'/>
       </message>
<message name='getDataResponse'>
     <part name='Result' type='xsd:float'/>
       </message>

<portType name='getDataPortType'>
    <operation name='getData'>
      <input message='tns:getDataRequest'/>
      <output message='tns:getDataResponse'/>
    </operation>
</portType>

<binding name='getDataBinding' type='tns:getDataPortType'>
    <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http'/>
     <operation name='getData'>
     <soap:operation soapAction='urn:getData#getData'/>
       <input>
         <soap:body use='encoded' namespace='urn:getData'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
        </input>
       <output>
      <soap:body use='encoded' namespace='urn:getData'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
     </output>
   </operation>
</binding>

<service name='getDataService'>
    <port name='getDataPort' binding='getDataBinding'>
    <soap:address location='http://hs19.iccs.bas.bg/nusoap'/>
```

```
 </port>
</service></definitions>
```

Anexo 2: Ejemplo de WorkFlow.



Anexo 3: Ejemplo de búsqueda en UDDI.

Ejemplo de mensaje enviado a UDDI para buscar servicios, en este caso se buscarán servicios de nombre *“Helloservice”*.

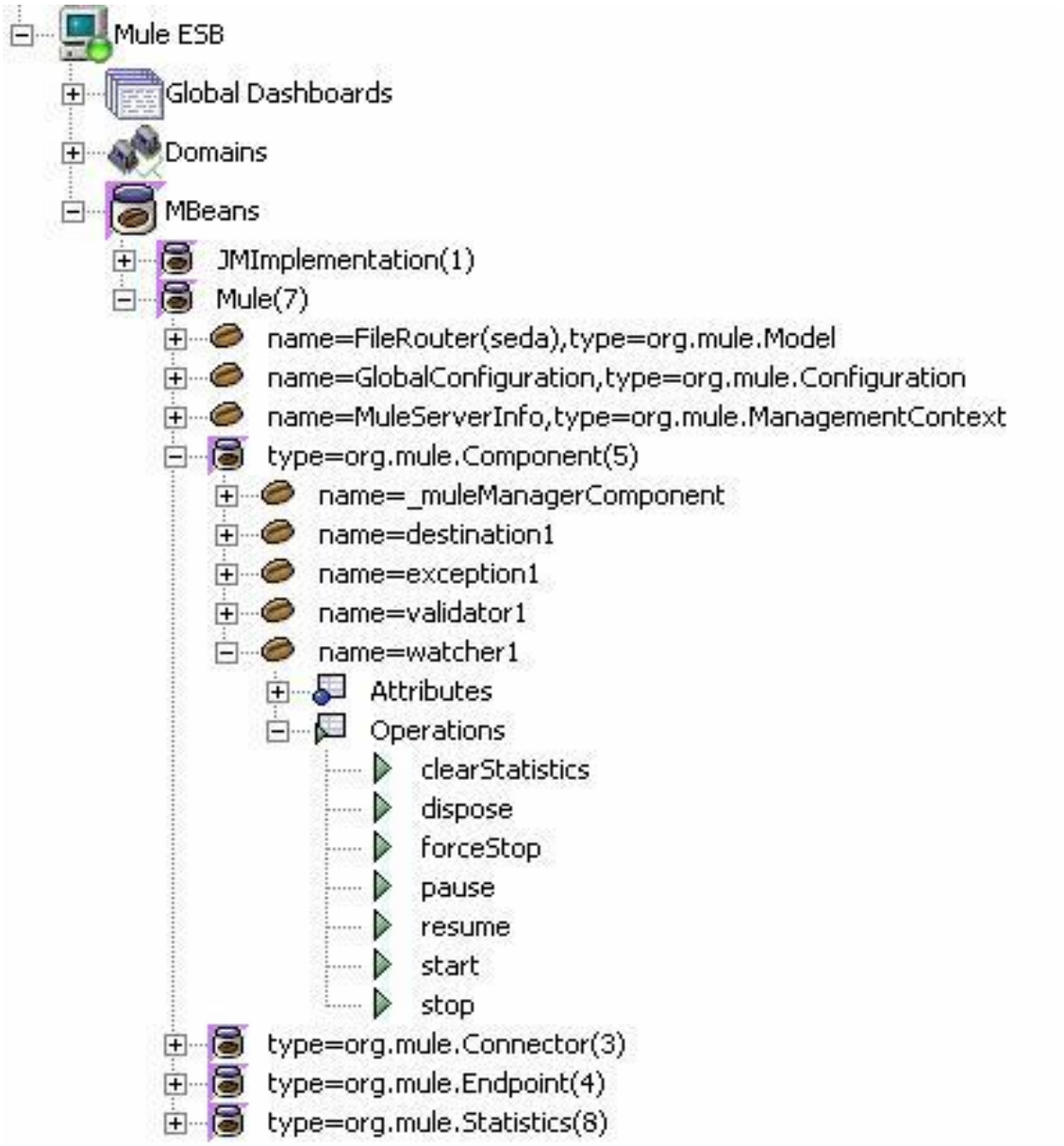
```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <find_service businessKey="*" generic="1.0" xmlns="urn:uddi-org:api">
      <name>helloservice</name>
    </find_service>
  </Body>
</Envelope>
```

Anexo 4: Ejemplo de registro en UDDI.

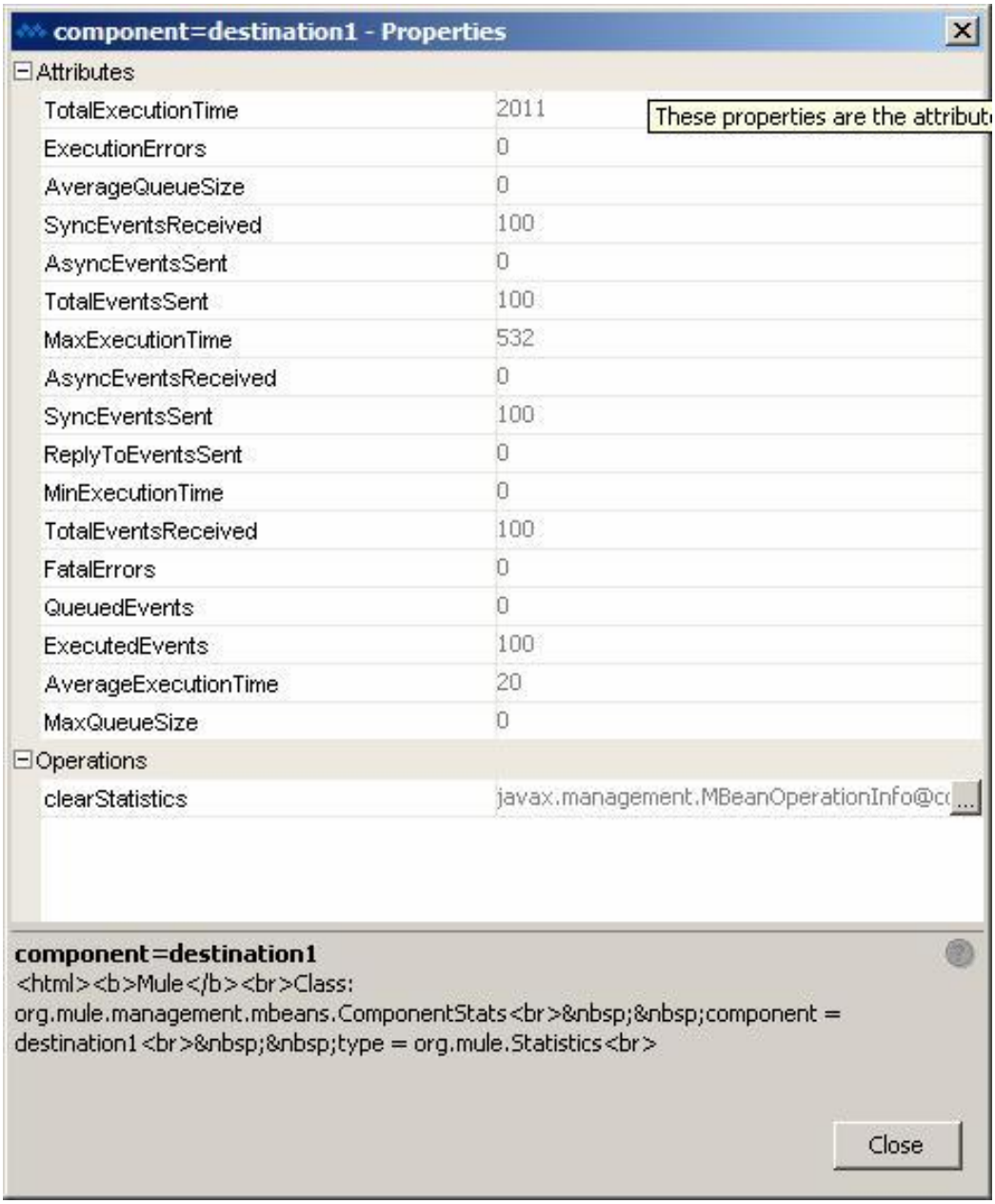
Publicación de un servicios llamado *"HelloService"* en UDDI.

```
<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 5: Vista gráfica de la administración de componentes de MuleESB.



Anexo 6: Vista gráfica del monitoreo de componentes de MuleESB.



Glosario de términos

API: (Application Programming Interface - Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ASP: (Active Server Pages): es una tecnología del lado servidor de Microsoft para páginas web generadas dinámicamente, que ha sido comercializada como un anexo a Internet Information Server (IIS). La tecnología ASP está estrechamente relacionada con el modelo tecnológico de su fabricante. Intenta ser solución para un modelo de programación rápida.

BPEL: Lenguaje de Ejecución de Procesos de Negocio, es un lenguaje estandarizado por OASIS para la composición de servicios web. Está desarrollado a partir de WSDL y XLANG, ambos lenguajes orientados a la descripción de servicios Web. Básicamente consiste en un lenguaje basado en XML diseñado para el control centralizado de la invocación de diferentes servicios Web, con cierta lógica de negocio añadida que ayudan a la programación en gran escala.

Broadcast: En castellano difusión, es un modo de transmisión de información donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

ETL: Son las siglas en inglés de Extraer, Transformar y Cargar (Extract, Transform and Load). Es el proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos, limpiarlos y cargarlos en otra base de datos para analizar, o en otro sistema operacional para apoyar un proceso de negocio.

FreeBSD: Es un sistema operativo libre para computadoras basado en las CPU de arquitectura Intel, incluyendo procesadores 386, 486 (versiones SX y DX), y Pentium.

FTP (File Transfer Protocol): Es un protocolo de transferencia de archivos entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor, de manera que desde un equipo cliente nos podemos conectar a un servidor para descargar archivos desde él o

para enviarle nuestros propios archivos independientemente del sistema operativo utilizado en cada equipo.

GNOME: Es un entorno de escritorio para sistemas operativos de tipo Unix bajo tecnología X Window. Forma parte oficial del proyecto GNU. Nació como una alternativa a KDE.

IDE: (Integrated Device Electronics "Dispositivo con electrónica integrada") o ATA (Advanced Technology Attachment), controla los dispositivos de almacenamiento masivo de datos, como los discos duros y ATAPI (Advanced Technology Attachment Packet Interface) y además añade dispositivos como las unidades CD-ROM.

JSP (JavaServer Pages): Es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo. Esta tecnología es un desarrollo de la compañía Sun Microsystems. La Especificación JSP 1.2 fue la primera que se liberó y en la actualidad está disponible la Especificación JSP 2.1.

KDE: Es un entorno de escritorio e infraestructura de desarrollo para sistemas Unix/Linux.

KPI (Key Performance Indicators): Son métricas financieras o no financieras, utilizadas para cuantificar objetivos que reflejan el rendimiento de una organización, y que generalmente se recogen en su plan estratégico.

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.

Middleware: Es un módulo intermedio que actúa como conductor entre sistemas permitiendo a cualquier usuario de sistemas comunicarse con varias fuentes de información u otros sistemas que se encuentran conectadas por una red.

NetBSD: Es un sistema operativo de la familia Unix (en sí no se le puede llamar "un Unix", ya que esta es una marca comercial de AT&T, pero se denomina como "sistema de tipo UNIX" o "derivado de UNIX"), open source y libre, y, a noviembre de 2006, disponible para más de 50 plataformas hardware.1 Su diseño y sus características avanzadas lo hacen ideal para multitud de aplicaciones. NetBSD ha surgido como resultado del esfuerzo de un gran número

de personas que tienen como meta producir un sistema operativo tipo Unix accesible y libremente distribuible.2

OpenBSD: Es un sistema operativo libre tipo Unix, multiplataforma, basado en 4.4BSD. Es un descendiente de NetBSD, con un foco especial en la seguridad y la criptografía.

Este sistema operativo, se concentra en la portabilidad, cumplimiento de normas y regulaciones, corrección, seguridad proactiva y criptografía integrada. OpenBSD incluye emulación de binarios para la mayoría de los programas de los sistemas SVR4 (Solaris), FreeBSD, GNU/Linux, BSD/OS, SunOS y HP-UX.

P2P: A grandes rasgos, una red informática entre iguales (en inglés peer-to-peer -que se traduciría de par a par- o de punto a punto, y más conocida como P2P) se refiere a una red que no tiene clientes ni servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red. Es una forma legal de compartir archivos de forma similar a como se hace en el email o mensajeros instantáneos solo que de una forma mas eficiente.

Paradigma: Un paradigma es —desde fines de la década de 1960— un modelo o patrón en cualquier disciplina científica u otro contexto epistemológico.

Perl (Lenguaje Práctico para la Extracción e Informe): Es un lenguaje de programación diseñado por Larry Wall creado en 1987. Perl toma características del C, del lenguaje interpretado shell (sh), AWK, sed, Lisp y, en un grado inferior, de muchos otros lenguajes de programación. Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y no tener ninguna de las limitaciones de los otros lenguajes de script.

PHP eZ publish: Manejador de contenidos.

PHP-Nuke: Es un sistema automatizado de noticias basado en la web y sistema de gestión de contenido basado en tecnologías PHP y MySQL. Para su instalación necesita un servidor web con soporte para PHP (por ejemplo Servidor HTTP Apache) así como una base de datos MySQL.

PHP BB: Es un conjunto gratuito de paquetes de código basados en el popular lenguaje de programación web PHP y lanzado bajo la Licencia pública general de GNU, cuya intención es la de proporcionar fácilmente, y con amplia posibilidad de personalización, un sistema de foros. Su nombre es por la abreviación de PHP Bulletin Board.

phpMyAdmin: Es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas webs, utilizando Internet.

PostgreSQL: Es un servidor de base de datos relacional orientada a objetos de software libre, liberado bajo la licencia BSD. Como muchos otros proyectos open source, el desarrollo de PostgreSQL no es manejado por una sola compañía sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Python: Es un lenguaje de programación creado por Guido van Rossum en el año 1990.1. Es comparado habitualmente con TCL, Perl, Scheme, Java y Ruby. En la actualidad Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation.

Ruby: Es un lenguaje de programación reflexivo y orientado a objetos (lenguaje interpretado). Combina una sintaxis inspirada en Python, Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU.

RUP: El Proceso Unificado Racional (Rational Unified Process en inglés) 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

Sistema Legacy: Es un sistema informático (equipos informáticos y/o aplicaciones) que se ha quedado anticuado y que continúa siendo utilizado por el usuario (típicamente una organización o empresa) y no se quiere o no puede ser reemplazado o actualizado de forma sencilla.

Secure Sockets Layer (SSL) y Transport Layer Security (TLS): Son protocolos criptográficos que proporcionan comunicaciones seguras en Internet. Existen pequeñas diferencias entre SSL 3.0 y TLS 1.0, pero el protocolo permanece sustancialmente igual. El término "SSL" según se usa aquí, se aplica a ambos protocolos a menos que el contexto indique lo contrario.

XSLT o Transformaciones XSL: Es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML.