UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 10

Título: Implementación de servicios para publicación, validación, custodia y transferencia de información en UDDI.

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

Autores:

Luis Manuel Rodríguez Hernández.

Ledián González Galindo.

Tutor:

Ing. Manuel Alejandro Gil Martín.

Ciudad de La Habana, Cuba

Junio de 2008

Declaración de Autoría

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

Para que así conste firmo la presente a los__ días del mes de__ del 2008.

_____ _____

Autor. Autor.

_____

Tutor.

Datos del Contacto

Tutor: Manuel Alejandro Gil Martin, graduado de Ingeniería Informática en el Instituto Superior Politécnico José Antonio Echeverría (CUJAE), Ciudad de La Habana, en el año 2005-2006. Actualmente se encuentra trabajando en la UCI como Especialista de la Dirección de Informatización y ocupa el cargo de jefe de Grupo de Plataformas, además de que imparte clases en la facultad 1. Su categoría docente es la de Instructor.

Agradecimientos

Al término de esta etapa de mi vida, quiero expresar un profundo agradecimiento a quienes con su ayuda, apoyo y comprensión me alentaron a lograr esta hermosa realidad.

A mis abuelos y a mis padres, porque sin escatimar esfuerzo alguno, han sacrificado gran parte de su vida para formarme y educarme.

A mis amigos por compartir tantos momentos de alegría y por haber llegado juntos a convertirnos en lo que somos hoy.

A todos los profesores que durante todos estos años fueron sembrando los conocimientos, que tanto han contribuido a lograr esta meta.

A mi tutor de tesis por guiarnos y porque gracias a su apoyo y consejo hemos llegado a realizar la más grande de nuestras metas.

A dios por brindarnos salud y acompañarnos en todo momento.

Ledián

A nuestra Revolución y a este Socialismo tan lindos, porque sin ellos, no hubiésemos podido conseguir jamás esta meta, debido a que soy de procedencia bien humilde.

A mis padres que en el transcurrir de mis días me han dado la mejor educación y me han guiado siempre por el buen camino, acompañándome en las alegrías y levantándome de las tristezas.

A mi adorada madre que es la persona más importante de mi vida, la que no me ha fallado nunca ni lo hará jamás, la que no se ha apartado de mí ni en un instante. A la persona por la cual respiro y que respira por mí.

A mi querido hermanito, que es la segunda persona que más me importa en la vida y me ha apoyado siempre de forma incondicional.

A todos los profesores que durante este período tan importante de mi vida me han formado y han dado lo mejor de si mismos para que pueda ser hoy lo que soy.

A mis amigos, con los que he compartido lo lindo y lo triste de esta Universidad y me han brindado apoyo y alegrías.

A Dios, porque sin ser religiosos, somos personas cultas y tenemos que agradecer a Dios por todo lo que tenemos.

Luis Manuel

Dedicatoria

Dedicado a mi linda familia, por haber confiado en mí, por acompañarme y por verme crecer siempre rodeado de apoyo y comprensión. A mi mamá por ser madre y amiga, por darme la oportunidad de saber lo lindo que es vivir. A mi querida abuela, a esa que me durmió tantas veces en sus brazos, que me cuidó, que me acompañó tras cada pesadilla, que me hizo tan feliz durante estos años.

Dedico especialmente, mi trabajo de diploma a:

Mi más grande ejemplo, a mi luz, mi guía, a esa persona que me regaló tantos momentos de alegría, que veló siempre por mí, sufriendo cada momento en que estuve enfermo, llorándome cuando estuve lejos y desbordando felicidad siempre que estoy cerca. Pillo, este es tu logro, este es tu trofeo, aunque sé que esto es insignificante, ante tanto cariño durante estos 23 años de tu vida. Felicidades mi querido abuelo "INGENIERO", tuya es mi alegría, mi felicidad, mi vida.

Ledián

Dedico este trabajo a todas las personas que de una forma u otra me apoyaron para que pudiera lograr hoy tan preciada meta.

Dedico especialmente este trabajo al ídolo de mi vida, mi madre. La persona que dicen que es muy importante en nuestras vidas, pero que en la mía de una forma resumida, lo es todo.

Luis Manuel

Resumen

El documento que tiene ante usted es el resultado de un estudio realizado en torno a UDDI (Universal Description, Discovery & Integration) y sus capacidades de extensión más allá del estándar actualmente aprobado. Se trata de un estándar de OASIS (Organization for the Advancement of Structured Information Standards) que permite gestionar un registro de servicios web. Actualmente existen en el mundo varias UDDIs debido al desarrollo científico técnico alcanzado. La mayoría de ellas las poseen las grandes empresas con el objetivo de tener una mayor organización en sus negocios; pero todas se encuentran bajo software propietario.

En el país y en la UCI (Universidad de las Ciencias Informáticas) como tal no existe ninguna UDDI capaz de satisfacer las necesidades en cuanto a la organización de los servicios web. Es por eso que se hace necesaria la creación de una UDDI. El tema de este trabajo es**: Implementación de servicios para publicación, validación, custodia y transferencia de información en UDDI**. Los objetivos concretos del trabajo son la implementación de las APIs que abarcan los servicios para publicación y validación, las cuales son: UDDI Publication API y UDDI Value Set API.

Se espera con este trabajo lograr un mayor entendimiento del mundo de las UDDIs y que las personas que accedan a él puedan aprender algo de este amplio campo, debido a que hasta el momento no se ha apreciado ningún trabajo relevante del tema.

Palabras Claves

OASIS, UDDI, publicación, validación, Publication API y Value Set API

Índice

Introducción

Los hombres del siglo XXI se encuentran inmersos en una sociedad en la que la tecnología de las comunicaciones y la informática han revolucionado el pensamiento y la forma de interactuar tanto entre personas, como entre personas y recursos informáticos, fundamentalmente a través de Internet, al permitir la utilización de muchas aplicaciones sobre una enorme red de redes sin requerir la instalación de componentes en el equipo del usuario. Solo se necesita una aplicación incorporada en la mayoría de los sistemas operativos: un navegador web. El éxito de Internet se basa en la utilización y amplísima aceptación por parte de toda la industria informática de estándares de red como la familia de protocolos TCP/IP (Transmission Control Protocol/Internet Protocol, Protocolo de Control de Transmisión/Protocolo Internet).

Recientemente surgió algo que se ha tornado una necesidad para las empresas que utilizan recursos informáticos, la arquitectura orientada a servicios (SOA). Fue desarrollada a finales de los noventa y no se trata de un software o un lenguaje de programación; sino que es un marco de trabajo conceptual que permite a las organizaciones unir los objetivos de negocio con la infraestructura de TI (Tecnología de Información) integrando los datos y la lógica de negocio de sus sistemas separados.

El objetivo de SOA es hacer que cada subsistema de una empresa presente sus capacidades a través de servicios adecuados, que permitan usar las capacidades de una manera homogénea. Esto evita crear interfaces para cada par de sistemas que requieran comunicación. Esta filosofía de crear servicios permite crear funciones que sean utilizables por distintos clientes; simplemente deben conocer la descripción del servicio para poder utilizarlo. La finalidad de SOA es lograr flexibilidad, reutilización y adaptabilidad para apoyar las demandas del negocio. En la mayoría de las definiciones de software estos servicios de los que se habla anteriormente son servicios web, aunque es posible implementar una SOA utilizando cualquier tecnología basada en servicios.

Los servicios web son una tecnología de integración de aplicaciones o interfaz, basada en estándares abiertos. Un servicio web no es más que un tipo de interfaz de un SI (Sistema Informático), concretamente se trata de una interfaz que ofrece las funcionalidades del SI en forma de servicios accesibles a través de los estándares abiertos utilizados en Internet como SOAP (Simple Object Access Protocol), XML (eXtensible Markup Language, Lenguaje de Marcado Ampliable o Extensible), WSDL (Web Services Description Language), UDDI o HTTP (Hyper Text Transfer Protocol).

Uno de los elementos clave adoptados es el XML como formato estándar de datos estructurados. XML es uno de los hitos tecnológicos más importantes de los últimos tiempos sobre el que existe infinidad de literatura, mucha gente discute y muy pocos lo entienden y lo abarcan completamente.

UDDI es un estándar para registrar y almacenar de forma estructurada los servicios web. La iniciativa de UDDI, surgió de la colaboración durante varios meses de representantes de Ariba, IBM (International Business Machines) y Microsoft iniciada en el verano del 2000, su soporte se ha expandido más allá de las tres compañías originales. Actualmente, el proyecto UDDI relaciona una comunidad de más de trescientos diez compañías. A través de UDDI, se puede publicar y descubrir información de una empresa y de sus servicios. 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. UDDI al igual que la mayor parte de los sistemas informáticos mantienen una relación de identidades personales (usuarios) asociadas normalmente con un perfil de seguridad, roles y permisos. En el mundo existen varias UDDIs, de acuerdo a las necesidades de las empresas, un ejemplo de estas es jUDDI, la cual es una UDDI desarrollada por el proyecto Apache, fue desarrollada en software libre, el lenguaje que utiliza es Java y fue implementada siguiendo las especificaciones 2.0 de OASIS. Su mayor dificultad es que no cuenta con una interfaz visual, es decir con una aplicación cliente.

Las tendencias que se concretan son aquellas que surgen en respuesta a necesidades. Pueden ser necesidades futuras, previstas por aquellos que son capaces de ver el próximo paso, o las que plantea hoy y ahora un usuario cada vez más exigente e insatisfecho. Una necesidad surgió aquí en la Universidad de Ciencias Informáticas (UCI), debido a que la UCI se encuentra en un crecimiento constante, el cual hace necesaria la búsqueda de opciones para un mejor manejo de información entre los principales sistemas de la universidad, por lo que la creación y utilización de servicios web se convierte en la mejor opción. En la UCI hay un considerable aumento de los servicios web disponibles para el desarrollo de aplicaciones. Esta gran aparición de Servicios Web trae varias dificultades como las que se muestran a continuación.

- ✓ Ubicación de los servicios web.
- ✓ Ordenamiento de los servicios web.
- ✓ Pérdida de servicios web.

- ✓ Mala Integración de los proyectos.
- ✓ Desconocimiento de la existencia de un servicio web dado, trayendo consigo la reimplementación innecesaria de este.
- ✓ Sitio para la publicación de los servicios web. Cuando un desarrollador realice un nuevo servicio web no tendrá donde publicarlo, provocando un bajo por ciento de reutilización de código.
- ✓ La no existencia de una herramienta de búsqueda. No se podrán realizar búsquedas por características, categorías, etc.
- ✓ Mala categorización de los servicios web. Al no contar con el servicio UDDI no se pueden agrupar los servicios web por categorías.

Es por ello que se hace necesario contar con algún tipo de herramienta que posibilite una eficiente manipulación y manejo de estos servicios. Desafortunadamente la UCI no cuenta con esta herramienta ni con la experiencia necesaria para esto. Por las razones expuestas anteriormente, se hace necesaria la búsqueda de una tecnología capaz de solucionar este problema. UDDI como alternativa para la solución de la problemática anterior, crea nuevas situaciones, como la creación de servicios que interactúen con ella. De aquí surge el **problema científico**, ¿cómo resolver la publicación y validación de información en UDDI sobre una plataforma libre en la UCI?

Este trabajo es antecedido por documentos que permiten conocer las principales características de UDDI. Trabajos que evidencian el comportamiento de los servicios de publicación y validación de información en UDDI como el del licenciado Pedro Espinosa, donde hace un estudio de las especificaciones técnicas de UDDI 3.02 de OASIS. Además de trabajos de diploma realizados en la UCI como: Estudio UDDI y Diseño de un catálogo de servicios Web basado en las especificaciones y normas de la UDDI para la plataforma de informatización en la UCI.

Se espera con este trabajo la creación y puesta en marcha de servicios para publicación y validación de información, lo cual se va a materializar con la implementación de las siguientes APIs: UDDI Publication API, UDDI Custody and Ownership Transfer API, UDDI Value Set Caching API y UDDI Value Set Validation API. El **objeto de estudio** de este trabajo se centra en las aplicaciones UDDI como sistemas gestionadores de servicios web para la UCI. El **campo de acción** por su parte está dirigido a las especificaciones técnicas de la versión UDDI 3.02 de OASIS. Con el objetivo de darle solución al problema antes planteado, durante todo el curso de este trabajo se plantea la siguiente **idea a defender**:

Si se realiza la implementación de servicios para publicación y validación de información, teniendo en cuenta los estándares internacionales y seleccionando las herramientas adecuadas, se puede contar con un sistema que dentro de su capa de lógica de negocios cuente con la implementación de las APIs: UDDI Publication API y UDDI Value Set API, brindando además la posibilidad de adicionar, eliminar y actualizar información de proveedores, así como los servicios Web que estos brindan, con todas las descripciones necesarias para su explotación, lo cual favorecerá el desarrollo y utilización de Servicios Web.

Teniendo en cuenta la arquitectura en tres capas que posee una aplicación UDDI, uno de los principales problemas para crear una UDDI es obtener una eficiente programación de las APIs que conforman la capa de lógica de negocios. La carencia en la UCI de la implementación de servicios para publicación y validación de información para conformar una UDDI confiable, segura y capaz de resolver los requerimientos de la arquitectura orientada a servicios constituye el principal motivo de esta investigación. Por lo que el **objetivo general** de este trabajo es implementar los servicios para la publicación y validación de información en UDDI de acuerdo a los estándares y especificaciones internacionales.

Dentro de los **objetivos específicos** del trabajo se encuentran los siguientes:

- ✓ Analizar el estado del arte de las APIs en UDDI.
- ✓ Analizar detalladamente las APIs UDDI planteadas en la especificación UDDI 3.02.
- ✓ Comparar las APIs UDDI en las diferentes versiones de UDDI existentes.
- ✓ Analizar el proceso de funcionamiento de cada servicio API en un registro UDDI.
- ✓ Desarrollar la implementación de la UDDI Publication API.
- ✓ Desarrollar la implementación de la UDDI Value Set API.

Con el propósito de guiar, controlar, evaluar y perfilar el trabajo hacia el alcance de los objetivos trazados, se definieron las siguientes **tareas**:

- ✓ Estudiar el estado del arte sobre las APIs en UDDI para determinar la situación de las APIs en las diferentes versiones de UDDI existentes.
- ✓ Realizar un estudio detallado de las APIs UDDI planteadas en la especificación UDDI 3.02 y escoger de estas, las APIs a incorporar en la UDDI a desarrollar.
- ✓ Comparar las APIs UDDI entre las diferentes versiones de UDDI existentes y determinar los cambios en las distintas versiones.

- ✓ Estudiar de forma completa y profunda el proceso de funcionamiento de cada servicio API en un registro UDDI para tener una mayor claridad a la hora de la implementación.

Durante el desarrollo de este trabajo se utilizaron métodos teóricos y métodos empíricos:

Analítico-sintético: se usa en el análisis de diversas metodologías y documentos de los cuales se extrajeron los rasgos distintivos relacionados con el desarrollo de sistemas UDDI.

Inductivo-deductivo: se utiliza para hacer un razonamiento del estado actual de UDDI y de todas las especificaciones técnicas existentes. Se llegó a un grupo de conocimientos, lo que permitirá llegar a conclusiones rápidas y avanzadas para comenzar la implementación de los servicios para publicación y validación de información.

Observación: con este método se logra un registro visual de lo que ocurre en una situación real, que es el caso de la UDDI de Windows Server, clasificando y consignando los hechos y acontecimientos del proceso.

Entrevista: para la recopilación de toda la información necesaria para el diseño de la aplicación, dígase flujo de información, datos de entrada, datos de salida, características de los procesos, etc.

El presente trabajo consta de tres capítulos, los cuales resumen la siguiente información:

Capítulo 1. FUNDAMENTACIÓN TEÓRICA: Descripción del objeto de estudio. Sistemas existentes vinculados al campo de acción. Tendencias y tecnologías actuales seleccionadas para el desarrollo de la aplicación y el por qué de su selección. Se explican los principales conceptos a tratar durante la investigación.

Capítulo 2. DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA: Se hace un análisis de códigos, componentes o módulos ya existentes con el objetivo de la reutilización de código. Cuenta con una descripción de los algoritmos no triviales utilizados, así como el análisis de la complejidad de los mismos y se seleccionan las estructuras de datos apropiadas para su implementación. Se hace una descripción de las clases que se utilizan.

Capítulo 3. VALIDACIÓN DE LA SOLUCIÓN PROPUESTA: Se realizan los tipos de prueba que son necesarios aplicar para poder afirmar que se cuenta con una aplicación de calidad. Estas pruebas pueden ser en tiempo de compilación (cuando se esta compilando el programa) o en tiempo de ejecución (cuando se ejecuta la aplicación). Se mostrarán además ejemplos de estos tipos de pruebas.

1

CAPÍTULO 1 Fundamentación Teórica

En este capítulo están contenidos los principales problemas que fundamentan la propuesta de solución, y los objetivos generales y específicos que se persiguen. Se brinda un enfoque general acerca de UDDI. Se describen y fundamentan las tecnologías actuales de desarrollo utilizadas para la implementación del sistema sobre las cuales se apoya la propuesta. Se abordan las distintas técnicas de programación que existen a nivel nacional e internacional, analizando su estado del arte. Se realiza un estudio profundo de los paradigmas de programación existentes para escoger los más apropiados para la solución.

1.1 Estado de las Técnicas de Programación a Nivel Internacional

Desde tiempos muy antiguos, las personas siempre seguían una guía o un ejemplo para construir algo, se tenía una idea o una noción de lo que se quería. El campo de la informática no es una excepción, para que esta rama haya alcanzado tanto desarrollo fue necesario que las cosas se hicieran de forma lógica. Hoy en día no se desarrolla ningún software o producto informático sino se tienen claras con antelación las metodologías y las técnicas a seguir.

Existen en el mundo múltiples técnicas o paradigmas de programación. Un paradigma de programación representa un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro [1]. Hay diferentes variantes a tener en cuenta a la hora de construir un software, cada una con sus metodologías y técnicas específicas, pero lo que si se puede asegurar es que actualmente no se desarrolla ningún software sino se definen con antelación alguna de estas variantes. En el estudio de los paradigmas de programación se han definido varias concepciones, la gran mayoría coincidiendo en destacar el imperativo, lógico, funcional y orientado a objetos como más ventajosos, lo que no quiere decir que no existan otros secundarios, pero no menos importantes.

1.2 Estado de las Técnicas de Programación en Cuba

Cuba no cuenta hoy con un alto desarrollo en la producción de software, pero se esta trabajando fuertemente por convertir al país en uno de los productores más importantes, un ejemplo de esto es la Universidad de las Ciencias Informáticas. En los pocos lugares en el país en los que se desarrolla software se hace siguiendo las normas y metodologías adecuadas, por lo que los paradigmas de programación juegan un papel fundamental.

1.3 Estado de las Técnicas de Programación en la UCI

La Universidad de las Ciencias Informáticas (UCI) es el lugar en el país con más desarrollo alcanzado en la rama de la informática. En la universidad se tiene un alto conocimiento de los paradigmas de programación, debido a que todo el software se hace siguiendo las técnicas y paradigmas adecuados. Se conocen y dominan todos los paradigmas existentes para poder determinar cuál usar a la hora de desarrollar un producto.

El paradigma de programación de uso más común es la Programación Orientada a Objetos por su gran importancia y aplicación en la modelación de objetos, lo cual es súper importante debido a que la mayoría de las cosas se pueden modelar como objetos. Conocer a profundidad los paradigmas más importantes es uno de los requisitos para formar los profesionales en la universidad, puesto que no se concibe un buen profesional que no domine esos conceptos. El continuo avance de las tecnologías y la informática provocará que un futuro aparezcan nuevos paradigmas de desarrollo. La UCI será de seguro parte de estos cambios.

1.4 Paradigmas de programación

Los paradigmas son marcos de referencia que imponen reglas sobre cómo se deben hacer las cosas, indican qué es válido dentro del paradigma y qué está fuera de sus límites [ídem a 1]. Un paradigma distinto implica nuevas reglas, elementos, límites y maneras de pensar, o sea implica un cambio. Los paradigmas pueden ser considerados como patrones de pensamiento para la resolución de problemas. Desde luego siempre teniendo en cuenta los lenguajes de programación, según el interés de estudio. Existen diferentes criterios a la hora de clasificar los paradigmas, según las clasificaciones establecidas por Floyd y Ambler, hay tres categorías fundamentales para su clasificación: [ídem a 1]

- ✓ Los que soportan técnicas de programación de bajo nivel, o sea los que su programación se acerca al funcionamiento de una computadora.
- ✓ Los que soportan métodos de diseño de algoritmos, llamados también de medio nivel.
- ✓ Los que soportan soluciones de programación de alto nivel, los cuales son más fáciles de aprender porque están formados por elementos de lenguajes naturales.

Existen varias clasificaciones para los paradigmas teniendo en cuenta estas categorías, estos se clasifican en:

- ✓ **Paradigmas de programación operacional o procedimental:** La característica fundamental de estos paradigmas es la secuencia computacional realizada etapa a etapa para resolver el problema.

Su mayor dificultad reside en determinar si el valor computado es una solución correcta del problema, por lo que se han desarrollado multitud de técnicas de depuración y verificación para probar la corrección de los problemas desarrollados basándose en este tipo de paradigmas. Estos pueden ser de dos tipos básicos [2]: con efecto de lado, dentro de los que se encuentran el paradigma imperativo [3] y el paradigma orientado a objetos, y sin efecto de lado.

- ✓ **Paradigmas Declarativos:** En este tipo, un programa se construye señalando hechos, reglas, restricciones, ecuaciones, transformaciones y otras propiedades derivadas del conjunto de valores que configuran la solución. A partir de esta información el sistema debe de proporcionar un esquema que incluya el orden de evaluación que compute una solución. Aquí no existe la descripción de las diferentes etapas a seguir para alcanzar una solución. En este grupo se encuentran: el funcional, el lógico y el de transformación [ídem a 3].
- ✓ **Paradigmas Demostrativos:** Cuando se programa bajo un paradigma demostrativo (también llamada programación por ejemplos), el programador no especifica procedimentalmente cómo construir una solución. En su lugar, presenta soluciones de problemas similares y permite al sistema que generalice una solución procedimental a partir de estas demostraciones [ídem a 2]. Estos pueden ser de tres tipos, de inducción, de redes de neuronas y genético.

Los paradigmas que se tuvieron en cuenta a la hora de darle solución al problema fueron el paradigma orientado a los servicios y el orientado a objetos.

1.4.1 Paradigma Orientado a Objetos

Un proyecto de software es algo realmente complejo. Las GUI (Graphical User Interface), el acceso transparente a datos y capacidad de trabajo en red, lo hacen más complejo aun. Para enfrentarse a esta complejidad nace la OPP (Object Oriented Programming). La programación orientada a objetos es una técnica o estilo de programación que utiliza objetos como bloque fundamental de construcción. [4]

1.4.1.2 Características más importantes de la OOP

- ✓ Abstracción: Significa extraer las propiedades esenciales de un objeto que lo distinguen de los demás tipos de objetos y proporciona fronteras conceptuales definidas respecto al punto de vista del observador. Es la capacidad para encapsular y aislar la información de diseño y ejecución [5].
- ✓ Encapsulamiento: Es el proceso de almacenar en un mismo compartimiento (una caja negra) los elementos de una abstracción (toda la información relacionada con un objeto) que constituyen su estructura y su comportamiento. Esta información permanece oculta tanto para los usuarios como para otros objetos y puede ser accedida solo mediante la ejecución de los métodos adecuados.

- ✓ Herencia: Es la propiedad que permite a los objetos construirse a partir de otros objetos. La clase base contiene todas las características comunes. Las sub-clases contienen las características de la clase base más las características particulares de la sub-clase. Si la sub-clase hereda características de una clase base, se trata de herencia simple. Si hereda de dos o más clases base, herencia múltiple.
- ✓ Polimorfismo: Literalmente significa "cualidad de tener más de una forma". En OPP, se refiere al hecho que una misma operación puede tener diferente comportamiento en diferentes objetos. En otras palabras, diferentes objetos reaccionan al mismo mensaje de modo diferente.

1.4.1.3 Ventajas de la OOP

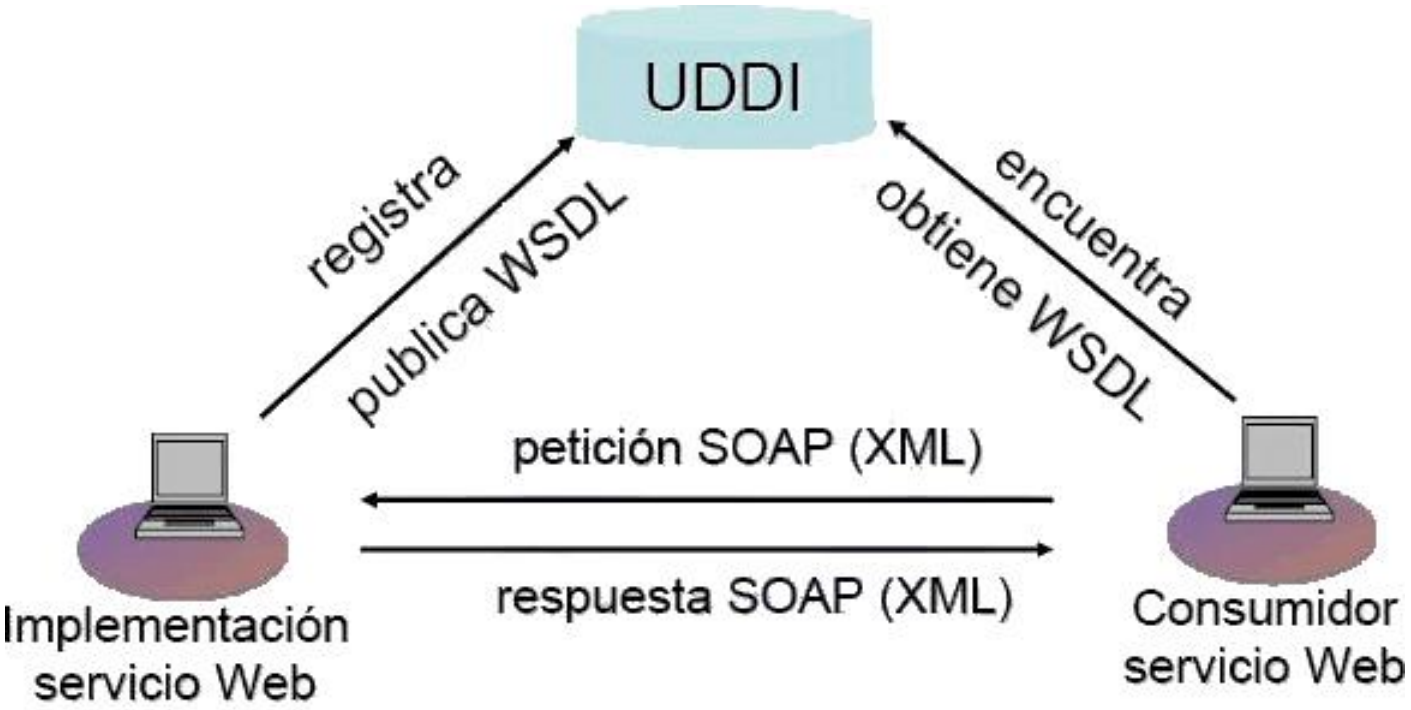
Modelos: La OPP permite realizar un modelo de sistema casi independientemente de los requisitos del proyecto. La razón es que en la OPP la jerarquía la establecen los datos, en cambio en la programación estructurada la jerarquía viene definida por los programas. Este cambio hace que los modelos se establezcan de forma similar al razonamiento humano y, por lo tanto, resulte más natural.

- ✓ Modularidad: Un programa es modular si se compone de módulos independientes y robustos. Esto permite la reutilización y facilita la verificación y depuración de los mismos. En OPP, los módulos están directamente relacionados con los objetos. Los objetos son módulos naturales ya que corresponden a una imagen lógica de la realidad [ídem a 5].
- ✓ Extensibilidad: Durante el desarrollo de sistemas, ocurre la aparición de nuevos requisitos, por eso es deseable que las herramientas de desarrollo permitan añadirlos sin modificar la estructura básica del diseño. En OPP es posible lograr esto siempre y cuando se hayan definido de forma adecuada la jerarquía de clases, los atributos y métodos.
- ✓ Eliminación de redundancia: En el desarrollo de sistemas se desea evitar la definición múltiple de datos y funciones comunes. En OPP esto se logra mediante la herencia (evita la definición múltiple de propiedades comunes a muchos objetos) y el polimorfismo (permite la modificación de métodos heredados). Solo hay que definir los atributos y los métodos en el antepasado más lejano que los comparte.
- ✓ Reutilización: La OPP proporciona un marco perfecto para la reutilización de las clases. El encapsulamiento y la modularidad nos permiten utilizar una y otra vez las mismas clases en aplicaciones distintas. En efecto, el aislamiento entre distintas clases significa que es posible añadir una nueva clase o un módulo nuevo (extensibilidad) sin afectar al resto de la aplicación.

1.4.2 Paradigma orientado a los servicios (SOA)

Es un estilo de arquitectura de sistemas informáticos para la creación y uso de los procesos de negocio a lo largo de su ciclo de vida [6]. SOA también define y proporciona una infraestructura que permite que diferentes aplicaciones puedan realizar un intercambio de datos y participar en los procesos de negocio. SOA separa las funciones en distintas unidades (servicios), que pueden ser distribuidos a través de una red y pueden ser combinados y reutilizados para crear aplicaciones de negocio. Estos servicios se comunican entre sí por pasar datos de un servicio a otro, o de una actividad de coordinación entre dos o más servicios. SOA puede también ser considerada como un estilo de la arquitectura de sistemas de información que permite la creación de aplicaciones que se construyen mediante la combinación de flexibilidad y servicios interoperables. La mayoría de las definiciones de SOA identifican la utilización de servicios Web empleando SOAP y WSDL en su implementación, lo cual no quiere decir que no se pueda implementar SOA utilizando cualquier tecnología basada en los servicios. A diferencia de las arquitecturas orientadas a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente inter-operables.

Figura I. Arquitectura Orientada a los Servicios



1.4.2.1 Diseño y desarrollo de 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 están hablando de un juego de servicios residentes en Internet o en una intranet, usando servicios web. Hay un juego de estándares de los que se habla ligados a los servicios web. Incluyen los siguientes [ídem a 6]:

- ✓ XML
- ✓ HTTP
- ✓ SOAP
- ✓ WSDL
- ✓ UDDI

Partes de la Arquitectura

Descripción de los servicios: WSDL	
Registro y búsqueda de servicios: UDDI	
Uso de los servicios: SOAP, HTTP, MIME	
Directorio: Publicar & encontrar Servicios:	**UDDI**
Inspección: Encontrar servicios en servidor:	**DISCO**
Descripción: Descripciones formales:	**WSDL**
Bits a transmitir: Interacciones de servicio:	**SOAP**
Formato universal de datos:	**XML**
Comunicaciones:	**Internet**

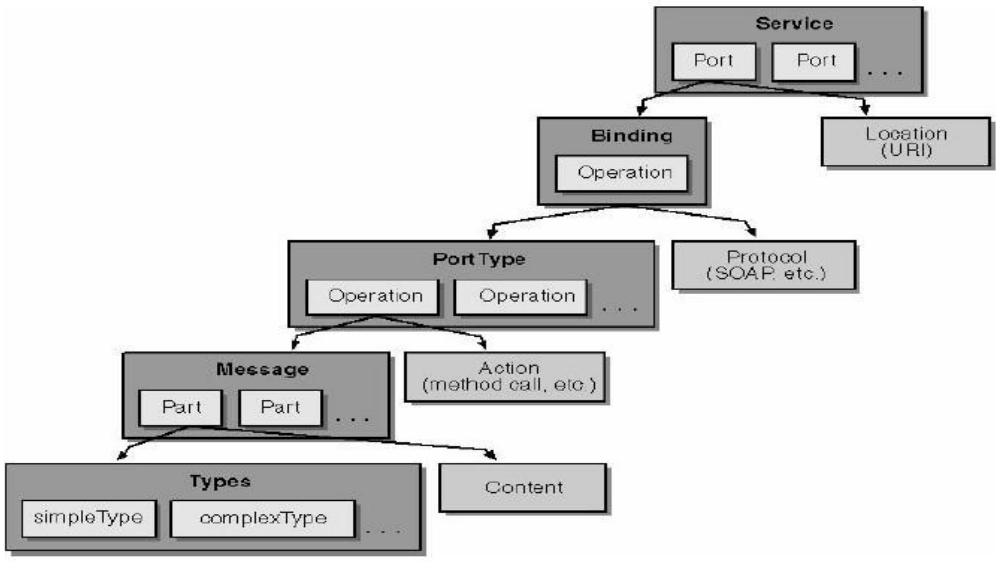
Hay que considerar, sin embargo, que un sistema SOA no necesariamente necesita utilizar estos estándares para ser "orientado a servicios" pero es altamente recomendable su uso.

1.5 WSDL

Esta basado en XML y se usa para describir servicios web [7]. Fue creado por Microsoft e IBM, es el lenguaje que usa UDDI, un sistema de registro estándar global de negocios. Describe la interfaz pública a los servicios Web. Describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje. Se usa a menudo en combinación con SOAP y XML Schema.

Un programa cliente que se conecta a un servicio web puede leer el WSDL para determinar qué funciones están disponibles en el servidor. Los tipos de datos especiales se incluyen en el archivo WSDL en forma de XML Schema. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL. A partir de WSDL se describen las funcionalidades del servicio Web, este logra aislar el lenguaje específico del servicio Web, logrando que se puedan distribuir los detalles del servicio Web en un lenguaje neutro, permitiendo que la implementación cliente sea implementada en distintos lenguajes y ambientes como: C++, Perl o Visual Basic.

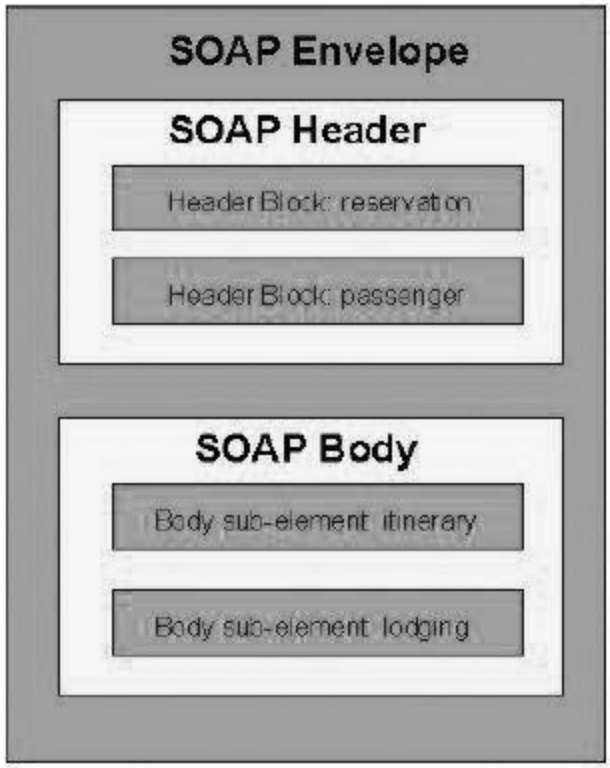
Figura II. Descripción de Servicios.



1.6 SOAP (Simple Object Access Protocol)

Es un protocolo estándar creado por Microsoft, IBM y otros, está actualmente bajo el auspicio de la W3C que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos utilizados en los servicios Web. SOAP usa el código fuente en XML. Esto es una ventaja ya que facilita su lectura por parte de humanos, pero también es un inconveniente dado que los mensajes resultantes son más largos. El intercambio de mensajes se realiza mediante tecnología de componentes (software componentry). El término Object en el nombre significa que se adhiere al paradigma de la programación orientada a objetos. SOAP funciona sobre cualquier protocolo de Internet, generalmente HTTP, que es el único registrado por la W3C. Tiene dos partes fundamentales: la cabecera (Header), es opcional y contiene metadatos sobre enrutamiento (routing), seguridad o transacciones. El desarrollo (Body) contiene la información principal, que se conoce como carga útil (payload). La carga útil se acoge a un XML Schema propio.

Figura III. Ejemplo de mensaje SOAP



1.7 XML

XML es un Lenguaje de Etiquetado Extensible muy simple, pero estricto, que juega un papel fundamental en el intercambio de una gran variedad de datos. Fue estandarizado por la W3C (Consorcio World Wide

Web). Ofrece una sintaxis sencilla para etiquetar documentos de forma legible para el hombre. Su formato es tan flexible que se utiliza en formatos de gráficos vectoriales (SVG), formatos de documentos multimedia (Open Document Format or Applications, OpenDocument), formatos para intercambio electrónico de datos (B2B), serialización de objetos, codificación de parámetros y métodos en los RPC (RPC-XML), etc. Los códigos XML se pueden crear y modificar mediante editores de texto. Algunas aplicaciones pueden manejar sólo algunos tipos de documentos XML, mientras que otras pueden interpretar cualquier tipo de XML [8].

1.7.1 Características de XML

- ✓ Existen distintas bibliotecas para su lectura, creación, manipulación, traducción en otros formatos, en distintas plataformas. XML añade "inteligencia" a los datos que se envían. Por sí solos, no tienen significado.
- ✓ XML añade metainformación, etiquetas autodescriptivas para cada parte del texto, de forma que el documento incluye el contenido y su descripción.
- ✓ Un conjunto de elementos XML relacionados forman un vocabulario.
- ✓ La instancia de un vocabulario se llama documento XML, que es el bloque fundamental de XML.
- ✓ Las distintas organizaciones que quieran intercambiar información con el formato XML pueden acordar el conjunto estándar de vocabularios, o proporcionar tecnologías para realizar la traducción de unos vocabularios a otros.
- ✓ No sólo los datos son textos, sino también las etiquetas. Esto permite que sea portable.
- ✓ Los vocabularios pueden declararse formalmente con un lenguaje de esquema XML.

1.8 HTTP

Apareció en los años 90 y nació la World Wide Web, una inmensa red de servidores HTTP que envían archivos por Internet y a la que se conoce por los diminutivos de "W3", "WWW" o "la Web". Mucha gente confunde la Web con Internet, siendo la primera tan sólo uno de los muchos servicios disponibles en Internet. Es un conjunto de estándares que les permite a los usuarios de la web intercambiar información. Es el método que se utiliza para transferir documentos desde el sistema donde se almacenan las páginas hasta los usuarios individuales.

1.8.1 Características de HTTP

- ✓ Protocolo de transferencia de hipertexto (HyperText Transfer Protocol, HTTP, RFC 2616).
- ✓ Utiliza protocolo de transporte TCP, y el puerto estándar del servicio es el 80.
- ✓ Es un protocolo orientado a transacciones y sin estados.

- ✓ Para poder direccionar un recurso en la web se utiliza el Localizador Universal de Recurso (URL, Universal Resource Locator, RFC 1738,1808).
- ✓ Posee mensajes en texto ASCII.

Tiene dos métodos básicos:

- ✓ GET: Solicitud para obtener la información identificada en la URL, obteniéndola en el cuerpo de la entidad.
- ✓ POST: Solicitud para admitir la entidad adjunta como subordinada a la URL indicada.

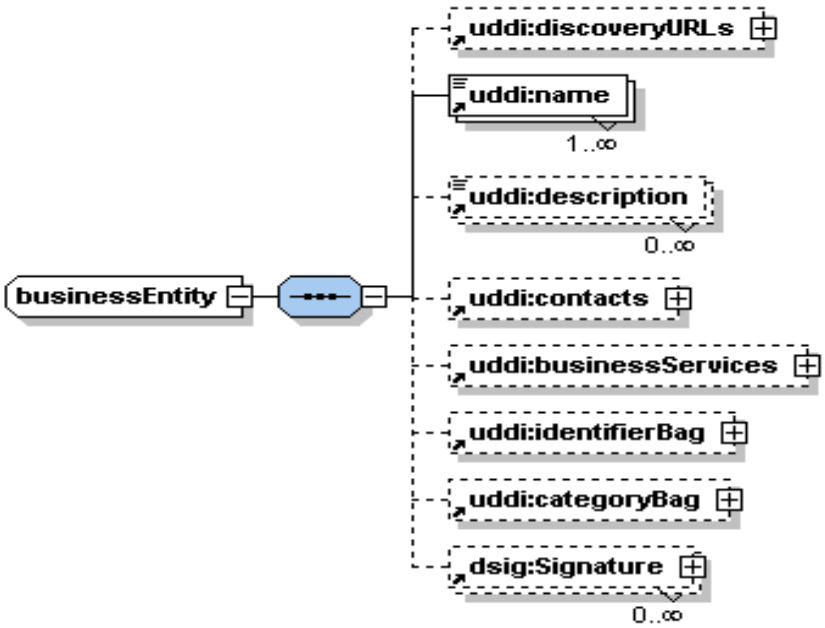
1.9 Estructuras de Datos UDDI

Las estructuras de datos conforman un modelo de información que almacena de forma persistente en el registro todos los elementos necesarios para su funcionamiento. Estas estructuras de datos se expresan en varios esquemas XML. Cada estructura de datos tiene un identificador único. Dentro de una UDDI se pueden identificar tres secciones: Sección Blanca, Sección Amarilla y Sección Verde.

1.9.1 BusinessEntity

Esta estructura describe a un proveedor de servicios Web y es la que almacena la sección Blanca de la que se hablaba anteriormente en el epígrafe 1.9 en las estructuras de datos UDDI. Contiene dentro de ella la estructura BusinessService [9].

Figura II. Estructura BusinessEntity

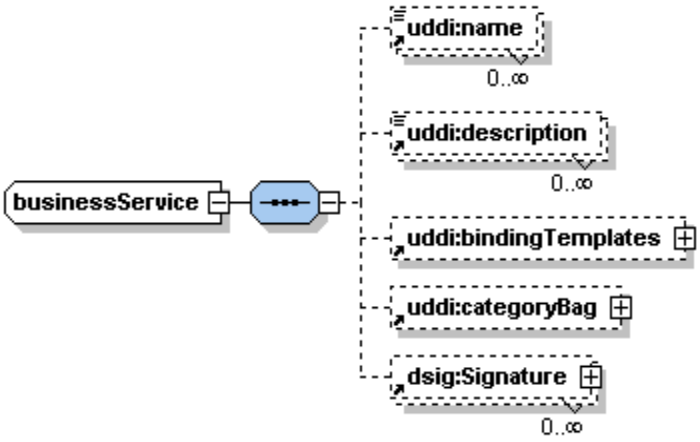


1.9.2 BusinessService

Esta estructura describe una familia de servicios Web ofrecidos por el proveedor descrito en el BussinessEntity y es la que almacena la sección Amarilla [ídem a 9].

Dentro de ella se encuentra contenida la estructura bindingTemplate.

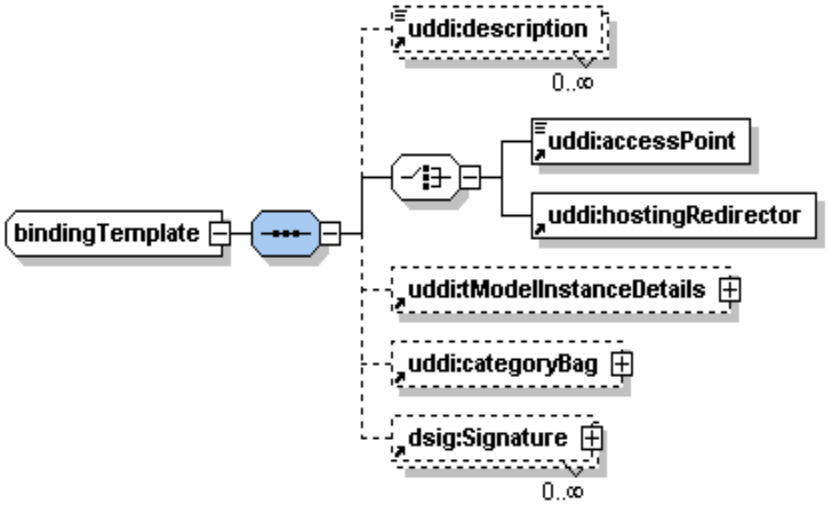
Figura III. Estructura BusinessService



1.9.3 BindingTemplate

Contiene las descripciones técnicas de los servicios web. Cada uno describe una instancia de un servicio web ofrecido normalmente a través de una URL, también describe el tipo de servicio web ofrecido utilizando modelos técnicos, parámetros de aplicaciones específicas y diferentes configuraciones. Entre esta estructura y el tModel, se encargan de almacenar la sección Verde [ídem a 9].

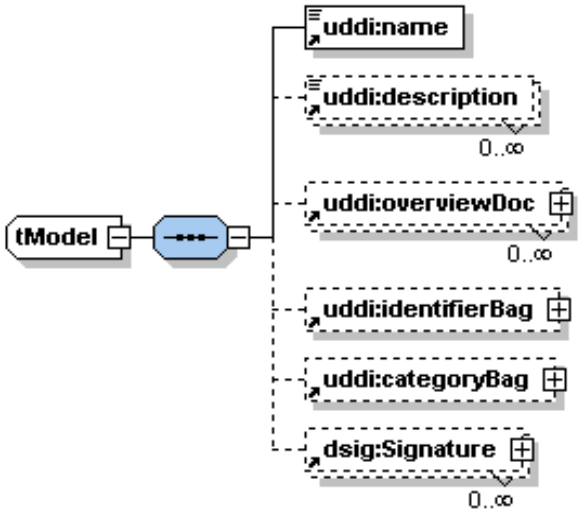
Figura IV. Estructura BindingTemplate



1.9.4 tModel

Las estructuras de tipo tModel describen modelos técnicos o metadatos reutilizables que pueden representar cualquier concepto como el tipo de servicio Web, algún protocolo utilizado por los servicios o un sistema de categorización [ídem a 9].

Figura V. Estructura tModel



1.10 Servicios Web

Un servicio web (Web service) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones [10]. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web.

1.10.1 Ventajas de los servicios Web

- ✓ Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- ✓ Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- ✓ Al apoyarse en HTTP, los servicios Web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.
- ✓ Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

- ✓ Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar.

1.10.2 Desventajas de los servicios Web

- ✓ Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).
- ✓ Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA, o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.
- ✓ Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

1.11 PHP

PHP (Hypertext Preprocessor) es un lenguaje script (no se compila para conseguir códigos máquina si no que existe un intérprete que lee el código y se encarga de ejecutar las instrucciones que contiene éste código), para el desarrollo de páginas web dinámicas del lado del servidor, cuyos fragmentos de código se intercalan fácilmente en páginas HTML, debido a esto, y a que es de Open Source (código abierto), es el más popular y extendido en la web. PHP es capaz de realizar determinadas acciones de una forma fácil y eficaz sin tener que generar programas programados en un lenguaje distinto al HTML. Esto se debe a que PHP ofrece un extenso conjunto de funciones para la explotación de bases de datos sin complicaciones.

PHP fue desarrollado originalmente por Rasmus Ledford en 1994 como un simple conjunto de scripts de Perl que permitía la interpretación de un número limitado de comandos. El sistema fue denominado **Personal Home Page** Tools y consiguió relativo éxito gracias a que otras personas pidieron a Rasmus que les permitiese utilizar sus programas en sus propias páginas. Cuando Rasmus tuvo la necesidad de crear páginas dinámicas que trabajasen con formularios, creó una serie de etiquetas a las que denominó “Form Interpreters”, y lo sacó al público con el nombre de PHP/FI en 1995. Luego salió la versión mejorada, llamada PHP/FI 2.0. Zeev Suraski y Andi Gutmans programaron el analizador sintáctico incluyendo nuevas funcionalidades como el soporte a nuevos protocolos de Internet y el soporte a la gran mayoría de las bases de datos comerciales, como MySQL y Postgre SQL, así como un módulo para Apache. Con estas mejoras surgió PHP3 en 1997. Este analizador define la sintaxis y semántica de la versión PHP3 y la siguiente: PHP4. La última versión es PHP5, que utiliza el motor Zend-2 y presenta mejoras significativas y un entorno de programación orientado a objetos mucho más completo, que permite que el PHP proporcione un alto rendimiento a las aplicaciones Web empresariales a nivel de las plataformas J2EE y .NET [11].

Una de sus características más potentes es su soporte para gran cantidad de bases de datos. Entre las que se pueden mencionar InterBase, mSQL, MySQL, Oracle, Informix, PostgreSQL, entre otras. PHP también ofrece la integración con varias bibliotecas externas, que dan al desarrollador la posibilidad de realizar cualquier tarea, desde generar documentos en pdf (Portable Document Format) hasta analizar código XML (eXtensible Markup Language) y últimamente también para la creación de otro tipo de programas incluyendo aplicaciones con interfaz gráfica usando la librería GTK+ [ídem a 11].

Es software libre, lo que implica menos costes y servidores más baratos que otras alternativas. Es muy rápido y su integración con la base de datos MySQL y el servidor Apache, le permite constituirse como una de las alternativas más atractivas del mercado [ídem a 11].

Es multiplataforma, funciona tanto para Unix (con Apache) como para Windows (con Microsoft Internet Information Server) de forma que el código que se haya creado para una de ellas no tiene porqué modificarse al pasar a la otra.

Su sintaxis está inspirada en C, ligeramente modificada para adaptarlo al entorno en el que trabaja, de modo que si se está familiarizado con esta sintaxis, le resultará muy fácil aprender PHP. Su librería estándar es realmente amplia, lo que permite reducir los llamados "costes ocultos", uno de los principales defectos de ASP (Active Server Pages).

Afortunadamente, lo nuevo de PHP 5 mejora muchas áreas en el lenguaje y su ejecución, como por ejemplo: [12]

- ✓ Programación orientada a objetos (OOP).
- ✓ MySQL.
- ✓ XML.
- ✓ Integración nativa con el Zend Engine.

Los diseñadores de PHP5 han realizado un cambio radical en el tratamiento de las variables objeto: en PHP5 todas las variables que nombran objetos son en realidad referencias. No hay que usar el operador '&' ni en las asignaciones, ni en el paso de parámetros que son objetos, ahorrándose con ello gran cantidad de potenciales errores [13].

La principal novedad en las clases de PHP5 es la inclusión de modificadores de control de acceso para implementar la encapsulación, piedra angular en la programación orientada a objetos de la que adolecía PHP4.

PHP5 introduce tres palabras clave (public, private y protected) que sustituyen a var en la definición de variables miembro, atributos de la clase, y que preceden a la definición de funciones miembro: métodos.

De acuerdo con todo lo expuesto anteriormente, PHP resulta mucho más favorecido, por tanto se toma como el adecuado para implementar la propuesta de sistema de este trabajo, particularmente PHP5 que es el más reciente y completo.

1.12 Servidor Web Apache

Se decidió usar servidor Apache por ser un software de código abierto que funciona sobre cualquier plataforma. Tiene capacidad para servir páginas tanto de contenido estático como de contenido dinámico a través de otras herramientas soportadas que facilitan la actualización de los contenidos mediante bases de datos, ficheros u otras fuentes de información. Es muy potente y altamente configurable.

Un servidor de páginas Web es un programa que permite acceder a páginas Web alojadas en un ordenador [ídem a 11]. Apache es el servidor web hecho por excelencia, su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa. La historia de Apache se remonta a febrero de 1995, donde empieza el proyecto del grupo Apache, el cual esta basado en el servidor Apache httpd de la aplicación original de NCSA. El desarrollo de esta aplicación original se estancó por algún tiempo tras la marcha de Rob McCool por lo que varios webmaster siguieron creando sus parches para sus servidores web hasta que se contactaron vía email para seguir en conjunto el mantenimiento del servidor web, fue ahí cuando formaron el grupo Apache.

Se decidió usar Apache como servidor web debido a las ventajas que proporciona, dentro de las cuales se pueden citar [ídem a 11]:

- ✓ Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- ✓ Apache es una tecnología gratuita de código fuente abierto. El hecho de ser gratuita es importante pero no tanto como que se trate de código fuente abierto. Esto le da una transparencia a este software de manera que si queremos ver que es lo que estamos instalando como servidor, lo podemos saber, sin ningún secreto, sin ninguna puerta trasera.
- ✓ Apache es un servidor altamente configurable de diseño modular. Es muy sencillo ampliar las capacidades del servidor Web Apache. Actualmente existen muchos módulos para Apache que son adaptables a este, y están ahí para que los instalemos cuando los necesitemos. Otra cosa importante es que cualquiera que posea una experiencia decente en la programación de C o Perl puede escribir un modulo para realizar una función determinada.
- ✓ Apache trabaja con gran cantidad de lenguajes Perl, PHP y otros lenguajes de script. Perl destaca en el mundo del script y Apache utiliza su parte del pastel de Perl tanto con soporte CGI como con soporte mod Perl. También trabaja con Java y páginas jsp. Teniendo todo el soporte que se necesita para tener páginas dinámicas.
- ✓ Apache te permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado script cuando ocurra un

error en concreto.

- ✓ Tiene una alta configurabilidad en la creación y gestión de logs. Apache permite la creación de ficheros de log a medida del administrador, de este modo puedes tener un mayor control sobre lo que sucede en tu servidor.

El servidor Apache es un software que esta estructurado en módulos, es decir, está dividido en muchas porciones de código que hacen referencia a diferentes aspectos o funcionalidades del servidor web. Esta modularidad es intencionada ya que la configuración de cada módulo se hace mediante la configuración de las directivas que están contenidas dentro del módulo. Los módulos del Apache se pueden clasificar en tres categorías [ídem a 11]:

- ✓ Módulos Base: Módulo con las funciones básicas del Apache.
- ✓ Módulos Multiproceso: Son los responsables de la unión con los puertos de la máquina, aceptando las peticiones y enviando a los hijos a atender a las peticiones.
- ✓ Módulos Adicionales: Cualquier otro módulo que le añada una funcionalidad al servidor.

Las funcionalidades más elementales se encuentran en el módulo base, siendo necesario un módulo multiproceso para manejar las peticiones. Se han diseñado varios módulos multiprocesos para cada uno de los sistemas operativos sobre los que se ejecuta el Apache, optimizando el rendimiento y rapidez del código. El resto de funcionalidades del servidor se consigue por medio de módulos adicionales que se pueden cargar. Para añadir un conjunto de utilidades al servidor, simplemente hay que añadirle un módulo, de forma que no es necesario volver a instalar el software.

1.13 Zend Studio

Se usó un editor de código fácil de trabajar y bien eficiente. Se esta hablando de Zend Studio, uno de los ambientes de desarrollo integrado o Integrated Development Environment (IDE) disponible para desarrolladores profesionales que agrupa todos los componentes de desarrollo necesarios para ciclo de desarrollo de aplicaciones PHP. A través de un amplio conjunto de herramientas de edición, depurado, análisis, optimización y bases de datos, Zend Studio acelera los ciclos de desarrollo y simplifica los proyectos complejos.

En este capítulo se exponen las condiciones y problemas que rodean el objeto de estudio; y a través de los conceptos y definiciones planteadas. Se evidencia la necesidad de implementar un software que permita la interoperabilidad, organización y publicación de los servicios Web en el contexto de

la arquitectura SOA. Para desarrollar el sistema se hace uso de la tecnología para la programación de páginas dinámicas, el lenguaje PHP5.

Capítulo 2 DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA

2

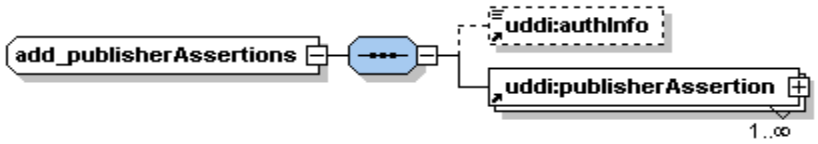
En el presente capítulo se plantea la solución propuesta para llevar a cabo la implementación de las APIs: UDDI Publication API y UDDI Value Set API, haciendo uso de sofisticados y eficientes métodos de programación. Se hace una valoración critica de las especificaciones de OASIS UDDI v3.02, como única base para la implementación. Cuenta con una descripción de los algoritmos no triviales utilizados, determinando la complejidad de los mismos. Se seleccionan las estructuras de datos apropiadas para la implementación de dichos algoritmos. Se hace una descripción detallada de las clases que se utilizan en la solución.

2.1 Valoración crítica del análisis propuesto en las especificaciones de UDDI3.02

La solución de este trabajo está basada en las especificaciones de UDDI3.02, por lo que el análisis a tener en cuenta es precisamente el que se describe en dichas especificaciones. En las especificaciones de UDDI 3.02 la UDDI Publication API cuenta con catorce requisitos funcionales, que pueden definirse como clases a la hora de la implementación:

Add_publisherAssertions: Esta clase es la que permite añadir las relaciones que se pueden establecer entre los proveedores de servicios. Tiene dos atributos: el **authInfo**, que es opcional según estas especificaciones, y es un mecanismo de seguridad, y el **publisherAssertion** que es el que tiene las relaciones que son añadidas.

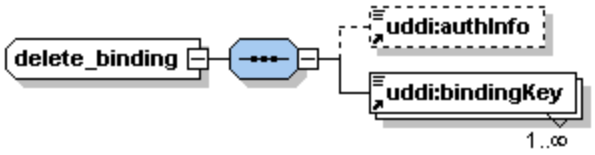
Figura VIII. Add_publisherAssertions



Delete_binding: Esta clase maneja la información necesaria para llevar a cabo la eliminación de una o más instancias de un binding template de un registro UDDI. Consta de dos atributos: el **authInfo**, que ya se

describía anteriormente y cumple la misma función en todas las clases, y el **bindingKey**, que esta contenido por uno o más valores uddiKey que representan instancias bindingTemplate específicas.

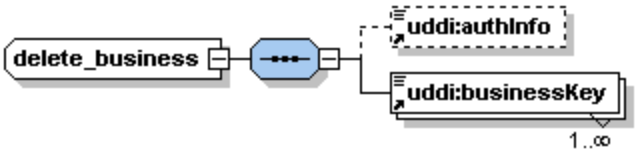
Figura IX. Delete_binding



Delete_business: Maneja y hace posible el modelado de la información para eliminar uno o más entidades de negocios o sea elementos **businessEntity** del registro UDDI. Tiene como atributos el **authInfo** y el **businessKey**, que es el que contiene uno o más valores uddiKey que representan instancias **businessEntity** específicas.

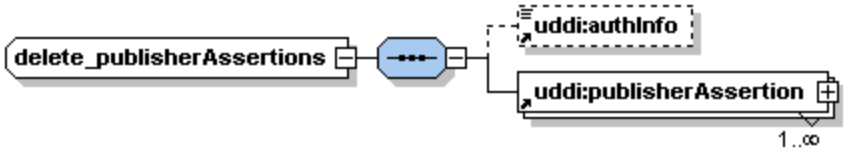
El registro UDDI debe eliminar permanentemente todo el contenido natural del elemento businessEntity, incluyendo cualquier dato de los businessService y bindingTemplate anidados del registro UDDI. El contenido natural no es más que todos los datos que se encuentran en el businessEntity registrado.

Figura X. Delete_business



Delete_publisherAssertions: Es la encargada de modelar lo necesario para eliminar una o más de las relaciones que se establecen en la clase Add_publisherAssertions. Tiene como atributos el **authInfo** y el **publisherAssertion**. El registro UDDI escanea la colección de assertion relacionadas con el publicador y borra cada una de las assertions que coincidan completamente con cada publisherAssertion pasada. Las assertions que son borradas aquí no aparecerán cuando se invoque la find_relatedBusinesses API.

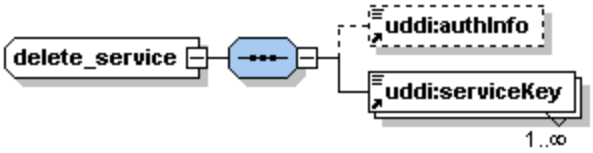
Figura XI. Delete_publisherAssertions



Delete_service: Es usada para el manejo de los datos necesarios, para llevar a cabo el borrado de uno o más elementos **businessService** del registro UDDI. Sus atributos son el **authInfo** y el **serviceKey** que esta formado por uno o más valores uddikey que representan instancias businessService específicas

Todos los datos del **bindindTemplate** contenido en estas estructuras **businessEntity** deben ser también eliminados.

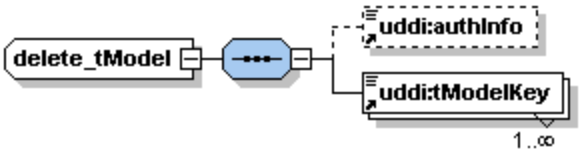
Figura XI. Delete_service



Delete_tModel: Modela lo imprescindible para borrar una o más estructuras tModel. Cuando es efectúa una llamada a la función que maneja esta clase, lo que hace es ocultar las estructuras tModel de los find_tModel, no las borra físicamente. Posteriormente se pueden crear nuevas referencias a estas estructuras ocultas por los publicadores que conocen sus llaves. Tiene dos atributos: el **authInfo** y el **tModelKey** que consiste en uno o más valores uddiKey que representan instancias tModel especificas

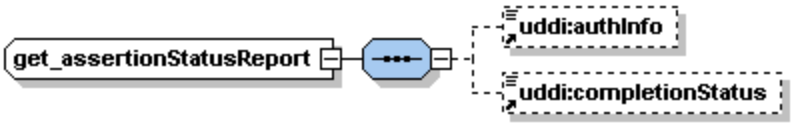
Como ya se decía anteriormente, cuando se ejecuta esta función se ocultan las estructuras tModel de los find_tModel, pero todas las demás APIs inquiry pueden contener referencias a estructuras tModel de tModelKeys eliminadas.

Figura XIII. Delete_tModel



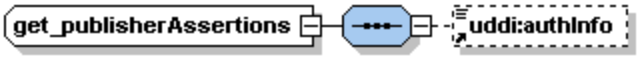
Get_assertionStatusReport: Esta clase maneja el comportamiento de la información que permite determinar el estado de las relaciones en las que participan proveedores de servicios. Usando esta llamada el publicador puede ver la posición de las relaciones que han hecho, así como ver las relaciones que otros han hecho que involucran estructuras businessEntity del publicador. Posee como atributos, el **authInfo** y el **completionStatus**, este último le permite al publicador restringir el resultado para aquellas relaciones que tienen el estado especificado. Los valores posibles son: **status: complete**, **status: toKey_incomplete** (devolvería sólo las relaciones a las que le falte la parte toKey), **status: fromKey_incomplete**, **status: both_incomplete** (se aplica sólo en el contexto de la suscripción UDDI).

Figura XIV. Get_assertionStatusReport



Get_publisherAssertions: Maneja la información correspondiente a una petición realizada al registro UDDI, con el objetivo de obtener todas las relaciones asociadas a un publicador, su único atributo es el parámetro opcional **authInfo** común a todas las funciones de las APIs de publicación, suscripción, consulta, etc., Que contiene un token de autenticación.

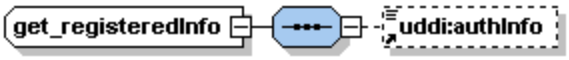
Figura XV. Get_publisherAssertions



Get_registeredInfo: Modela todo lo imprescindible para obtener un listado abreviado de los proveedores de servicio y los modelos técnicos controlados por un publicador. El publicador es el usuario autenticado y dado de alta como tal en el registro que invoca la function.

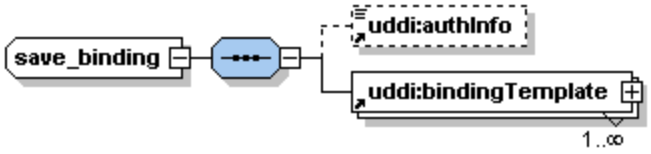
Tiene un atributo **infoSelection** obligatorio que representa la cantidad de **tModels** que deben devolverse, la lista de valores es: all, visible (devuelve los que no se hayan eliminado lógicamente) y hidden (devuelve los que hayan sufrido un borrado lógico del registro). Su otro atributo es el **authInfo.**

Figura XVI. Get_registeredInfo



Save_binding: Maneja y controla la información que será almacenada y/o actualiza la secuencia de bindingTemplate pasada como parámetro, incluyendo su relación con los servicios correspondientes. Tiene dos atributos: el **authInfo** y el **bindingTemplate** que contiene una o más estructuras bindingTemplate. Cada bindingTemplate pasado debe contener un valor serviceKey que corresponda con un businessService registrado, controlado por el mismo publicador. Si el bindingKey dentro del bindingTemplate no existe o es pasado con un valor vacío, esto significa que el bindingTemplate es insertado por primera vez. Cuando esto ocurre el nodo debe generar automáticamente una nueva llave para el bindingTemplate que está sin una llave asociada.

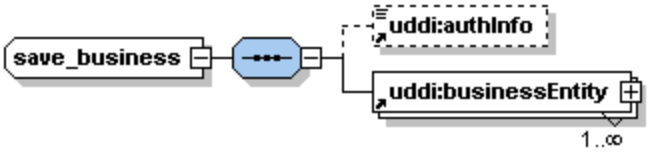
Figura XVII. Save_binding



Save_business: Modela los datos de una secuencia de businessEntity para salvarlos o actualizarlos en el registro. Esta clase tiene el mayor alcance de todas las clases save_xxx y puede ser usada para hacer cambios extensos a la información publicada en uno o más elementos businessEntity controlados por un individuo. Tiene dos atributos: el **authInfo** y el arreglo **businessEntity** que contiene una o más estructuras businessEntity. Si cualquiera de los valores uddiKey dentro del businessEntity no existe o es pasado con un valor vacío, esto significa que el businessEntity es insertado por primera vez. Cuando esto ocurre el

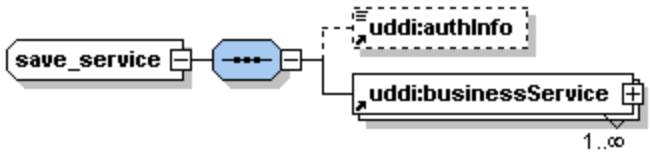
nodo debe generar automáticamente una nueva llave para el businessEntity que está sin una llave asociada.

Figura XVIII. Save_business



Save_service: Esta clase controla los datos para almacenar y/o actualizar una secuencia de businessService. Cada **businessService** puede ser firmado y debe tener la llave del publicador asignada. Tiene dos atributos: el **authInfo** y el **businessService** que contiene uno o más elementos **businessService**. Con el propósito de mejorar la forma de actualización, este dato puede ser obtenido de forma avanzada usando la get_serviceDetail API o por cualquier otro medio. Cada businessService pasado debe contener un valor businessKey que corresponda al businessEntity registrado por el mismo publicador. Si cualquiera de los valores uddiKey dentro del businessService no existe o es pasado con un valor vacío, esto significa que el businessService es insertado por primera vez. Cuando esto ocurre el nodo debe generar automáticamente una nueva llave para el businessService que está sin una llave asociada.

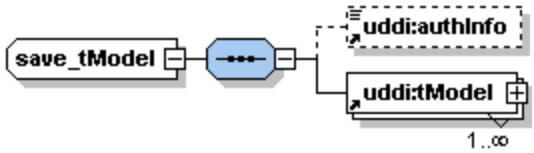
Figura XIX. Save_service



Save_tModel: Maneja y modela la información que quiere ser almacenada y/o actualizada de una secuencia de tMode. Los tModels pueden ser firmados y pueden ser salvados con la llave asignada al publicador, incluyendo aquellos tModels que establecen la partición de dominio de las llaves asignadas a los publicadores, los cuales son conocidos como tModels generadores de llaves de dominio. Tiene dos atributos: el **authInfo** y el **tModel** que contiene uno o más elementos tModel. Con el propósito de mejorar la forma de actualización, este dato puede ser obtenido de forma avanzada usando la get_tModel o por cualquier otro medio. Si cualquiera de los valores uddiKey dentro del tModel no existe o es pasado con un valor vacío, esto significa que el tModel es insertado por primera vez. Cuando esto ocurre el nodo debe generar automáticamente un nuevo **tModelKey** para este dato. Esta clase ejecuta una actualización a los

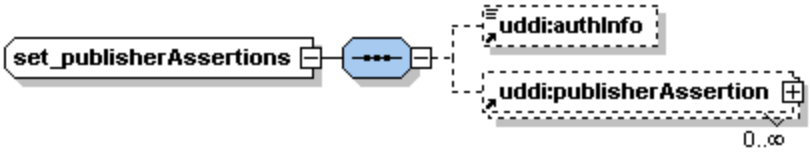
datos existentes registrados cuando el valor tModelKey tiene un valor uddiKey que corresponde con un dato ya registrado.

Figura XX. Save_tModel



Set_publisherAssertions: Esta clase gestiona las relaciones de un publicador. Tiene dos atributos: el **authInfo** y el **publisherAssertion**, que puede ser una secuencia vacía y que sustituirá a todas las existentes controladas por el mismo publicador. Si no se especifica, se borrarán las existentes. Devuelve una estructura publisherAssertions con la secuencia de relaciones actuales del publicador.

Figura XXI. Set_publisherAssertions



2.2 Descripción de los algoritmos no triviales a implementar

En la actualidad existen diversa cantidad de algoritmos, hay algoritmos para búsquedas, algoritmos basados en las matemáticas, algoritmos genéticos, etc. De forma general un algoritmo no es más que una secuencia de pasos finitos que se deben seguir para realizar una tarea determinada.

Uno de los procesos o algoritmos que se llevan a cabo en UDDI Publication API es el de salvar negocio (save_business). En este proceso se almacena y/o actualiza la secuencia de businessEntity pasada como parámetro. Devuelve una estructura businessDetail con la información registrada o actualizada. Esta estructura es una secuencia de businessEntity. Como en el resto de funciones save_xxx, al procesar esta llamada, el registro UDDI posiblemente tendrá que generar claves, en el proceso de alta, si el publicador no las indica en la llamada. En las actualizaciones, las claves tienen que indicarse.

Esta función puede mover servicios de un proveedor a otro. Se puede borrar información con esta función, cuando la información registrada sea diferente a la proporcionada en esta llamada, no se borra del nodo de custodia, pero la información que no se especifique del proveedor de servicios que ya exista en el registro será borrada, al actualizarse. Cuando un servicio contenido en un proveedor apunta, a través de su “clave

ajena" a otro proveedor diferente, se anota en el registro como una proyección del servicio. Cuando se almacena un proveedor con servicios proyectados, se ignora todo su contenido, salvo las claves del servicio y del proveedor de servicios.

2.2.1 Análisis de complejidad del algoritmo

Para determinar la complejidad del algoritmo que se describe en el epígrafe 2.2, lo que se hará es calcular la complejidad ciclomática del mismo, para hacer dicho cálculo es necesario primero tener el código o el diseño del algoritmo, posteriormente enmarcar cada instrucción del código con un número, que representa cada lugar del camino que puede seguir la secuencia del algoritmo, a continuación se muestra el código con sus instrucciones enmarcadas.

```
public function execute ($regObject)
{

        $request = $regObject;                          1
        $generic = $request->getGeneric ();             1
        $authInfo = $request->getAuthInfo ();
        $businessVector = $request->getBusinessEntityVector ();     1
        $uploadRegVector = $request->getUploadRegisterVector ();    1

        $uuidgen = new DefaultUUIDGen ();   1

        if (($uploadRegVector != null) && (sizeof($uploadRegVector) > 0))    2
        {
                throw new UnsupportedException("save_business: ". "UploadRegistry is not supported.");     3
        }

        $dataStore = new JDBCDataStore ();              4

        $dataStore->beginTrans ();              4
        $publisher = $this->getPublisher ($authInfo, $dataStore);           4
        $publisherID = $publisher->getPublisherID ();               4
        $authorizedName = $publisher->getName ();                   4

        for ($i=0; $i< sizeof($businessVector); $i++)       5
        {
                $business = $businessVector[$i];        6
                $businessKey = $business->getBusinessKey ();  6

                if (($businessKey! = null) && (strlen($businessKey) > 0) && (!$dataStore->isValidBusinessKey ($businessKey)))   7
                {
                throw new InvalidKeyPassedException("save_business: ". "businessKey=".$businessKey);       8
                }

                if (($businessKey! = null) && (strlen($businessKey) > 0) && (!$dataStore-
                >isBusinessPublisher($businessKey,$publisherID)))       9
                {
                throw new UserMismatchException("save_business: "."userID=". $publisherID+", "."businessKey=". $businessKey);  10
                }

                $categoryBag = $business->getCategoryBag ();  11

                 if ($categoryBag != null)                  12
                {
                  $keyedRefVector = $categoryBag->getKeyedReferenceVector();    13
                            if ($keyedRefVector != null)                        14
                             {
```

```
                        $vectorSize = sizeof($keyedRefVector);              15
                        if ($vectorSize > 0)                                16
                        {
                            for ($j=0; $j<$vectorSize; $j++)                17
                            {
                                    $keyedRef = $keyedRefVector[$j];        18
                                    $key = $keyedRef->getTModelKey();       18

                                    if (($key == null) || (strlen(trim($key)) == 0))    19
                                    {
                                    $keyedRef->setTModelKey($tModel->GENERAL_KEYWORDS_TMODEL_KEY);    20
                                    }
                            }       21
                        }
                    }
            } 22

}

for ($i=0; $i<sizeof ($businessVector); $i++)     23
{
            $business = $businessVector [$i];     24
            $businessKey = $business->getBusinessKey ();          24
            if (($businessKey != null) && (strlen($businessKey) > 0))          25
            {
                    $dataStore->deleteBusiness($businessKey);     26
            }
            else
            {
                    $business->setBusinessKey ($uuidgen->uuidgenprefix ());    27
            }

            $this->addBusinessEntityDiscoveryURL ($business);     28
            $business->setAuthorizedName ($authorizedName);       28

            $contacts = $business->getContacts ();          28
            if (($contacts == null) || ($contacts->getContactVector () == null) || (sizeof($contacts->getContactVector())==0))    29
            {
                    $contact = new Contact();               30
                    $contact->setPersonNameValue ($publisher->getName());      30
                    $contact->setUseType ("publisher");     30
                    $email = $publisher->getEmailAddress();       30
                    if ($email != null)             31
                    {
                            $contact->addEmail(new Email($email, "email"));    32
                    }

            $business->addContact ($contact);               33                                                              }

$dataStore->saveBusiness ($business, $publisherID);       34

}

            $dataStore->commit ();      35
            $detail = new BusinessDetail ();      35
$detail->setGeneric ($generic);         35
$detail->setTruncated (false);          35
$detail->setBusinessEntityVector ($businessVector);     35

if ($dataStore! = null)         36
{
            $dataStore->release ();     37
}

return $detail;     38
```

```
}
```

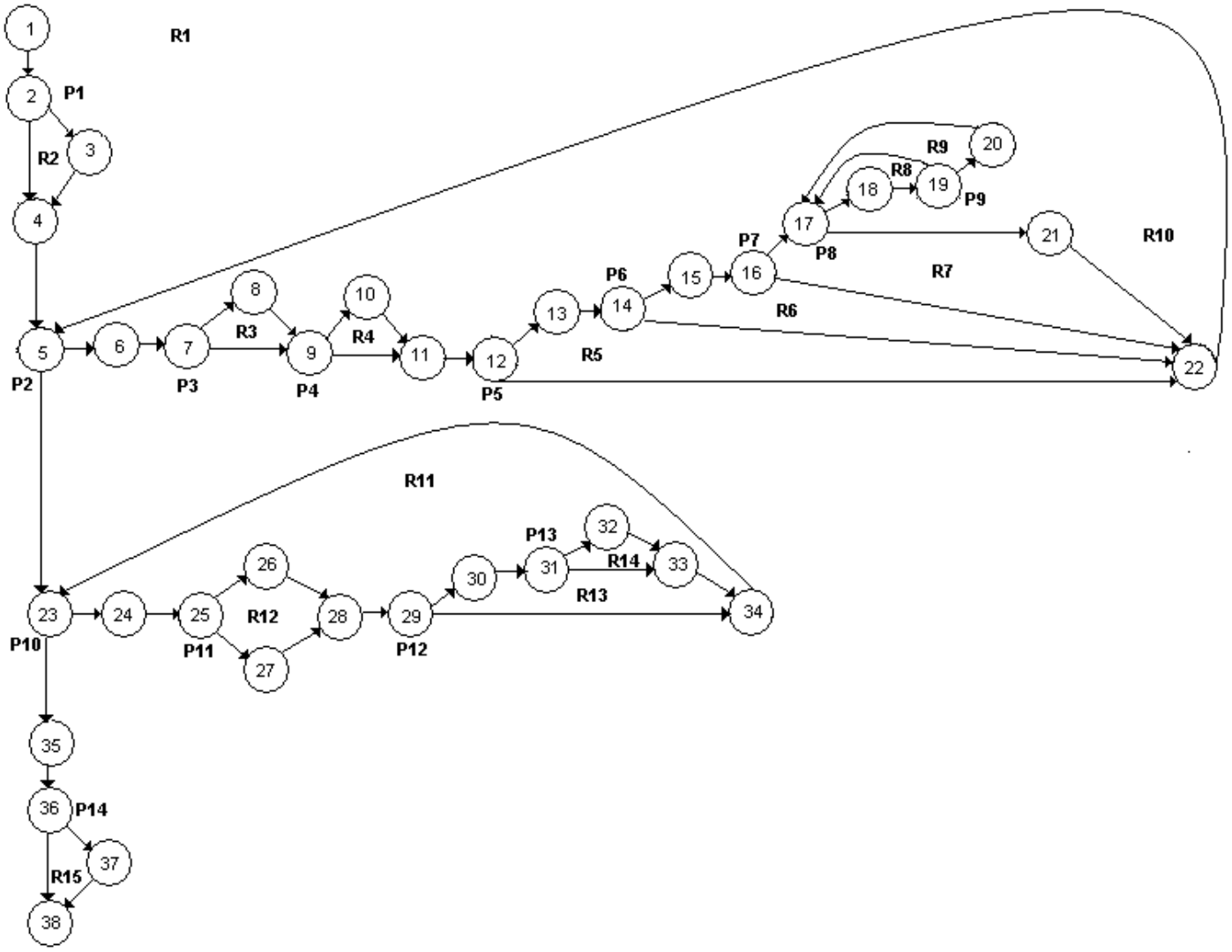
Después de este paso, es necesario representar el grafo de flujo asociado, en el cual se representan distintos componentes como es el caso de:

Nodo: Se le llama a los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.

Aristas: Están constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aun cuando el nodo no representa la sentencia de un procedimiento.

Regiones: Las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran siendo la cantidad de regiones equivalente a la cantidad de caminos independientes del conjunto básico de un programa. Ver figura XXII.

Figura XXII Grafo de flujo del algoritmo.



Seguidamente a la construcción del grafo de flujo se procede a efectuar el cálculo de la complejidad ciclomática del código, el cálculo es necesario efectuarlo mediante tres vías o fórmulas que para concluir que el cálculo fue correcto es necesario que por las tres vías el resultado sea el mismo, las fórmulas para calcular son las siguientes:

1. V (G) = R número de regiones del grafo de flujo.

2. V (G) = A - N + 2

Donde: **A** es el número de aristas del grafo y **N** es el número de nodos.

3. V (G) = *P* + 1

Donde: ***P*** es el número de nodos predicado (nodo del cual salen varias aristas) contenidos en el grafo ***G***.

Total de regiones R = 15

Aristas A = 51

Nodos N = 38

Nodos P = 14

V (G) = **R**

V (G) = **15**

V (G) = *A - N + 2*

V (G) = *51 - 38 + 2*

V (G) = 15

V (G) = *P* + 1

V (G) = *14 +1*

V (G) = 15

Realizado el cálculo por las 3 vías necesarias se llega a la conclusión que el algoritmo presentado anteriormente tiene un complejidad ciclomática de 15 que da la visión de que existen a lo sumo 15 caminos lógicos por donde recorrer el algoritmo.

2.3 Estructuras de datos apropiadas para la implementación de estos algoritmos.

En la realización de un script en PHP en múltiples ocasiones existen variables que tienen información similar y se procesan de forma semejante. Para ello PHP (y otros lenguajes) poseen un elemento denominado array. Un array es un conjunto de variables agrupadas bajo un único nombre. Cada variable dentro de la matriz se denomina elemento. Dentro de la misma matriz pueden existir variables de diferentes tipos y no es necesario que sean todas del mismo tipo.

Una matriz en PHP es en realidad un mapa ordenado. Un mapa es un tipo de datos que asocia valores con claves. Este tipo es optimizado en varias formas, de modo que puede usarlo como una matriz real, o una lista (vector), tabla asociativa (caso particular de implementación de un mapa), diccionario, colección, pila, cola y probablemente más. Ya que puede tener otra matriz PHP como valor, es realmente fácil simular árboles.

2.4 Descripción de las nuevas clases u operaciones necesarias

AddPublisherAssertions	
Atributo	**Tipo**
$authInfo $publisherAssertionVector	No tiene
Responsabilidades	
Nombre:	Descripción:
AddPublisherAssertions setAuthInfo_in_AddPublisherAssertions addPublisherAssertion_in_AddPublisherAssertions setPublisherAssertionVector_in_AddPublisherAssertions	Crea el objeto **AddPublisherAssertions.** Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro. Recibe un objeto **PublisherAssertion** y lo añade al arreglo contenido en el atributo **$publisherAssertionVector**. Recibe un arreglo y lo asigna al atributo **$publisherAssertionVector.**

Tabla 1 Descripción de la clase AddPublisherAssertions

AddPublisherAssertionsHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **AddPublisherAssertionsHandler**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (AddPublisherAssertions soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **AddPublisherAssertions** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 2 Descripción de la clase AddPublisherAssertionsHandler

AddPublisherAssertionsFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct execute	Crea el objeto **AddPublisherAssertionsFunction.** Este método recibe un objeto (AddPublisherAssertions este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente llama la función encargada de controlar el envió de la información a la base de datos UDDI.

Tabla 3 Descripción de la clase AddPublisherAssertionsFunction

DeleteBinding	
Atributo	**Tipo**
$authInfo $bindingKeyVector	No tiene
Responsabilidades	
Nombre:	Descripción:
DeleteBinding	Crea el objeto **DeleteBinding.**
setAuthInfo_in_DeleteBinding	Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.
addBindingKey_in_DeleteBinding	Recibe una cadena con el **BindingKey** o objeto de este tipo, y añade el **BindingKey** al arreglo contenido en el atributo **$bindingKeyVector**.
setBindingKeyVector_in_DeleteBinding	Recibe un arreglo y lo asigna al atributo **$bindingKeyVector.**

Tabla 4 Descripción de la clase DeleteBinding

DeleteBindingHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **DeleteBindingHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (DeleteBinding soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **DeleteBinding** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 5 Descripción de la clase DeleteBindingHandler

DeleteBindingFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **DeleteBindingFunction.**
execute	Este método recibe un objeto (**DeleteBinding** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente llama la función encargada de controlar el envió de la información a la base de datos UDDI.

Tabla 6 Descripción de la clase DeleteBindingFunction

DeleteBusiness	
Atributo	**Tipo**
$authInfo $businessKeyVector	No tiene
Responsabilidades	
Nombre:	Descripción:
DeleteBusiness	Crea el objeto **DeleteBusiness.**
setAuthInfo_in_DeleteBusiness	Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.
addBusinessKey	Recibe una cadena con el **BusinessKey** o objeto de este tipo, y añade el **BusinessKey** al arreglo contenido en el atributo **$businessKeyVector.**
setBusinessKeyVector_in_DeleteBusiness	Recibe un arreglo y lo asigna al atributo **$businessKeyVector.**

Tabla 7 Descripción de la clase DeleteBusiness

DeleteBusinessHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **DeleteBusinessHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (DeleteBusiness soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **DeleteBusiness** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 8 Descripción de la clase DeleteBusinessHandler

DeleteBusinessFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct execute	Crea el objeto **DeleteBusinessFunction.** Este método recibe un objeto (DeleteBusiness este objeto es retornado por la función unmarshal de la clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente llama la función encargada de controlar el envió de la información a la base de datos UDDI.

Tabla 9 Descripción de la clase DeleteBusinessFunction

DeletePublisherAssertions	
Atributo	**Tipo**
$authInfo $publisherAssertionVector	No tiene
Responsabilidades	
Nombre:	Descripción:
DeletePublisherAssertions	Crea el objeto **DeleteBusiness.**
setAuthInfo_in_DeletePublisherAssertions	Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.
addPublisherAssertion_in_DeletePublisherAssertions	Recibe un objeto **PublisherAssertion** y lo añade arreglo contenido en el atributo **$publisherAssertionVector**
setPublisherAssertionVector_in_DeletePublisherAssertions	Recibe un arreglo y lo asigna al atributo **$publisherAssertionVector.**

Tabla 10 Descripción de la clase DeletePublisherAssertions

DeletePublisherAssertionsHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **DeletePublisherAssertionsHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (DeletePublisherAssertions soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **DeletePublisherAssertions** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 11 Descripción de la clase DeletePublisherAssertionsHandler

DeletePublisherAssertionsFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **DeletePublisherAssertionsFunction.**
execute	Este método recibe un objeto (DeletePublisherAssertions este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente llama la función encargada de controlar el envió de la información a la base de datos UDDI.

Tabla 12 Descripción de la Clase DeletePublisherAssertionsFunction

DeleteService	
Atributo	**Tipo**
$authInfo $serviceKeyVector	No tiene
Responsabilidades	
Nombre:	Descripción:
DeleteService	Crea el objeto **DeleteService.**
setAuthInfo_in_DeleteService	Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.
addServiceKey_in_DeleteService	Recibe una cadena con el **ServiceKey** o objeto de este tipo, y añade el **ServiceKey** al arreglo contenido en el atributo **$serviceKeyVector.**
setServiceKeyVector_in_DeleteService	Recibe un arreglo y lo asigna al atributo **$serviceKeyVector.**

Tabla 13 Descripción de la clase DeleteService

DeleteServiceHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **DeleteServiceHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**DeleteService** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **DeleteService** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 14 Descripción de la clase DeleteServiceHandler

DeleteServiceFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct execute	Crea el objeto **DeleteServiceFunction.** Este método recibe un objeto (**DeleteService** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente llama la función encargada de controlar el envió de la información a la base de datos UDDI.

Tabla 15 Descripción de la clase DeleteServiceFunction

DeleteTModel	
Atributo	**Tipo**
$authInfo $tModelKeyVector	No tiene
Responsabilidades	
Nombre:	Descripción:
DeleteTModel	Crea el objeto **DeleteTModel.**
setAuthInfo_in_DeleteTModel	Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.
addTModelKey_in_DeleteTModel	Recibe una cadena con el **TModelKey** o objeto de este tipo, y añade el **TModelKey** al arreglo contenido en el atributo **$tModelKeyVector.**
setTModelKeyVector_in_DeleteTModel	Recibe un arreglo y lo asigna al atributo **$tModelKeyVector.**

Tabla 16 Descripción de la clase DeleteTModel

DeleteTModelHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **DeleteTModelHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**DeleteTModel** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **DeleteTModel** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 17 Descripción de la clase DeleteTModelHandler

DeleteTModelFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **DeleteTModelFunction.**
execute	Este método recibe un objeto (**DeleteTModel** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente llama la función encargada de controlar el envío de la información a la base de datos UDDI.

Tabla 18 Descripción de la clase DeleteTModelFunction

GetAssertionStatusReport	
Atributo	**Tipo**
$STATUS_COMPLETE $STATUS_TOKEY_INCOMPLETE $STATUS_FROMKEY_INCOMPLETE $authInfo $completionStatus	No tiene
Responsabilidades	
Nombre:	Descripción:
GetAssertionStatusReport	Crea el objeto **GetAssertionStatusReport.**
setAuthInfo_in_GetAssertionStatusReport	Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.
setCompletionStatus	Recibe una cadena con el **status** o un objeto de tipo **CompletionStatus**, y asigna el valor del **status** al atributo **$completionStatus.**
getCompletionStatus	Devuelve el atributo **$completionStatus.**

Tabla 19 Descripción de la clase GetAssertionStatusReport

GetAssertionStatusReportHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **GetAssertionStatusReportHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**GetAssertionStatusReport** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **GetAssertionStatusReport** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 20 Descripción de la clase GetAssertionStatusReportHandler

GetAssertionStatusReportFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct execute	Crea el objeto **GetAssertionStatusReportFunction.** Este método recibe un objeto (**GetAssertionStatusReport** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente crea un objeto **AssertionStatusReport** y lo retorna.

Tabla 21 Descripción de la clase GetAssertionStatusReportFunction

GetPublisherAssertions	
Atributo	**Tipo**
$authInfo	No tiene
Responsabilidades	
Nombre:	Descripción:
GetPublisherAssertions setAuthInfo_in_GetPublisherAssertions	Crea el objeto **GetPublisherAssertions.** Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.

Tabla 22 Descripción de la clase GetPublisherAssertions

GetPublisherAssertionsHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **GetPublisherAssertionsHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**GetPublisherAssertions** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **GetPublisherAssertions** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 23 Descripción de la clase GetPublisherAssertionsHandler

GetPublisherAssertionsFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **GetPublisherAssertionsFunction.**
execute	Este método recibe un objeto (**GetPublisherAssertions** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente crea un objeto **PublisherAssertions** y lo retorna.

Tabla 24 Descripción de la clase GetPublisherAssertionsFunction

GetRegisteredInfo	
Atributo	**Tipo**
$authInfo	
Responsabilidades	
Nombre:	Descripción:
GetRegisteredInfo setAuthInfo_in_GetRegisteredInfo	Crea el objeto **GetRegisteredInfo.** Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.

Tabla 25 Descripción de la clase GetRegisteredInfo

GetRegisteredInfoHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **GetRegisteredInfoHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**GetRegisteredInfo** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **GetRegisteredInfo** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 26 Descripción de la clase GetRegisteredInfoHandler

GetRegisteredInfoFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct execute	Crea el objeto **GetRegisteredInfoFunction.** Este método recibe un objeto (**GetRegisteredInfo** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente crea un objeto **RegisteredInfo** y lo retorna.

Tabla 27 Descripción de la clase GetRegisteredInfoFunction

SaveBinding	
Atributo	**Tipo**
$authInfo $bindingTemplateVector	
Responsabilidades	
Nombre:	Descripción:
SaveBinding	Crea el objeto **SaveBinding.**
setAuthInfo_in_SaveBinding	Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.
addBindingTemplate	Recibe un objeto **BindingTemplate** y lo añade al arreglo contenido en el atributo **$bindingTemplateVector.**
setBindingTemplateVector_in_SaveBinding	Recibe un arreglo y lo asigna al atributo **$bindingTemplateVector.**

Tabla 28 Descripción de la clase SaveBinding

SaveBindingHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SaveBindingHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**SaveBinding** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **SaveBinding** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 29 Descripción de la clase SaveBindingHandler

SaveBindingFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SaveBindingFunction.**
execute	Este método recibe un objeto (**SaveBinding** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, llama la función encargada de controlar el envió de la información a la base de datos UDDI, finalmente crea un objeto **BindingDetail** y lo devuelve.

Tabla 30 Descripción de la clase SaveBindingFunction

SaveBusiness	
Atributo	**Tipo**
$authInfo $businessVector $uploadRegisterVector	
Responsabilidades	
Nombre:	Descripción:
SaveBusiness	Crea el objeto **SaveBusiness.**
setAuthInfo_in_SaveBusiness	Asigna al atributo $authInfo un objeto (AuthInfo) pasado por parámetro.
addBusinessEntity_in_SaveBusiness	Recibe un objeto **BusinessEntity** y lo añade al arreglo **$businessVector.**
setBusinessEntityVector_in_SaveBusiness	Recibe un arreglo y lo asigna al atributo **$businessVector.**
addUploadRegister_in_SaveBusiness	Recibe un objeto **UploadRegister** y lo añade al arreglo **$uploadRegisterVector.**
setUploadRegisterVector_in_SaveBusiness	Recibe un arreglo y lo asigna al atributo **$uploadRegisterVector.**

Tabla 31 Descripción de la clase SaveBusiness

SaveBusinessHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SaveBusinessHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**SaveBusiness** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **SaveBusiness** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 32 Descripción de la clase SaveBusinessHandler

SaveBusinessFunction	
Controladora	
Atributo	**Tipo**
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SaveBusinessFunction.**
execute	Este método recibe un objeto (**SaveBusiness** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, llama la función encargada de controlar el envió de la información a la base de datos UDDI, finalmente crea un objeto **BusinessDetail** y lo devuelve.
addBusinessEntityDiscoveryURL	Método para agregar el URL donde puede ser visto el **BusinessEntity.**

Tabla 33 Descripción de la clase SaveBusinessFunction

SaveService	
Atributo	**Tipo**
$authInfo $serviceVector	
Responsabilidades	
Nombre:	Descripción:
SaveService	Crea el objeto **SaveService.**
setAuthInfo_in_SaveService	Asigna al atributo $authInfo un objeto (**AuthInfo**) pasado por parámetro.
addBusinessService_in_SaveService	Recibe un objeto **BusinessService** y lo añade al arreglo contenido en el atributo **$serviceVector.**
setServiceVector_in_SaveService	Recibe un arreglo y lo asigna al atributo **$serviceVector.**

Tabla 34 Descripción de la clase SaveService

SaveServiceHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SaveServiceHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**SaveService** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **SaveService** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 35 Descripción de la clase SaveServiceHandler

SaveServiceFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SaveServiceFunction.**
execute	Este método recibe un objeto (**SaveService** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, llama la función encargada de controlar el envió de la información a la base de datos UDDI, finalmente crea un objeto **ServiceDetail** y lo devuelve.

Tabla 36 Descripción de la clase SaveServiceFunction

SaveTModel	
Atributo	**Tipo**
$authInfo $tModelVector $uploadRegisterVector	
Responsabilidades	
Nombre:	Descripción:
SaveTModel	Crea el objeto **SaveTModel.**
setAuthInfo_in_SaveTModel	Asigna al atributo $authInfo un objeto (**AuthInfo**) pasado por parámetro.
addTModel_in_SaveTModel	Recibe un objeto **Tmodel** y lo añade al arreglo **$serviceVector.**
setTModelVector_in_SaveTModel	Recibe un arreglo y lo asigna al atributo **$tModelVector.**
addUploadRegister_in_SaveTModel	Recibe un objeto **UploadRegister** y lo añade al arreglo **$serviceVector.**
setUploadRegisterVector_in_SaveTModel	Recibe un arreglo y lo asigna al atributo **$uploadRegisterVector.**

Tabla 37 Descripción de la clase SaveService

SaveTModelHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SaveTModelHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**SaveTModel** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **SaveTModel** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 38 Descripción de la clase SaveTModelHandler

SaveTModelFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct execute	Crea el objeto **SaveTModelFunction.** Este método recibe un objeto (**SaveTModel** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, llama la función encargada de controlar el envió de la información a la base de datos UDDI, finalmente crea un objeto **TModelDetail** y lo devuelve.

Tabla 39 Descripción de la clase SaveTModelFunction

SetPublisherAssertions	
Atributo	**Tipo**
$authInfo $publisherAssertionVector	
Responsabilidades	
Nombre:	Descripción:
SetPublisherAssertions	Crea el objeto **SetPublisherAssertions.**
setAuthInfo_in_SetPublisherAssertions	Asigna al atributo $authInfo un objeto (**AuthInfo**) pasado por parámetro.
addPublisherAssertion_in_SetPublisherAssertions	Recibe un objeto **PublisherAssertion** y lo añade al arreglo contenido en el atributo **$publisherAssertionVector.**
setPublisherAssertionVector_in_SetPublisherAssertions	Recibe un arreglo y lo asigna al atributo **$publisherAssertionVector.**

Tabla 40 Descripción de la clase SetPublisherAssertions

SetPublisherAssertionsHandler	
Controladora	
Atributo	**Tipo**
$TAG_NAME $maker	
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SetPublisherAssertionsHandler.**
unmarshal	Método que recibe un elemento XML, con la estructura de un mensaje SOAP de petición de la función (**SetPublisherAssertions** soportada por UDDI) al cual le da tratamiento sacando toda su información, para modelarla luego en un objeto, en este caso de tipo **SetPublisherAssertions** el cual es retornado al final.
marshal	Este método recibe un objeto de respuesta y un elemento XML, crea otro elemento XML con la información que extrae del objeto y por ultimo añade el nuevo XML como hijo del que le pasaron por parámetro.

Tabla 41 Descripción de la clase SetPublisherAssertionsHandler

SetPublisherAssertionsFunction	
Controladora	
Atributo	**Tipo**
No tiene	No tiene
Responsabilidades	
Nombre:	Descripción:
__construct	Crea el objeto **SetPublisherAssertionsFunction.**
execute	Este método recibe un objeto (**SetPublisherAssertions** este objeto es retornado por la función unmarshal del clase anteriormente descrita), realiza todo el trámite correspondiente para sacar la información de este objeto, y finalmente crea un objeto **PublisherAssertions** y lo retorna.

Tabla 42 Descripción de la clase SetPublisherAssertionsFunction

El trabajo desarrollado en este capítulo fue de gran importancia para una mejor comprensión del sistema que se implementó, debido a que en él se hace una descripción minuciosa de las clases fundamentales utilizadas en la implementación, lo que posibilitará futuros mantenimientos o cambios al software.

La descripción de uno de los algoritmos no triviales fue de gran ayuda para dar a conocer en ese caso como fue resuelto el problema de salvar negocio en la UDDI Publication API y no menos importante fue la explicación del uso de las estructuras de datos usadas, debido a los beneficios que brindan las estructuras **array** y **matriz** en PHP.

Capítulo 3 VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

Cuando se esta desarrollando cualquier aplicación es inevitable cometer errores, errores que si no se les da tratamiento pueden afectar el correcto funcionamiento del sistema. Una forma de comprobar que el trabajo que se está realizando esta libre de errores es testeando el código o las interfaces.

En este capítulo se abordará fundamentalmente sobre los tipos de pruebas que es necesario realizarle a un sistema para que sea probado ya sea en tiempo de compilación (cuando se esta compilando el programa) o en tiempo de ejecución (cuando se ejecuta la aplicación). Se mostrarán además ejemplos de estos tipos de pruebas.

3.1 Descripción de las pruebas.

Los “test de unidades” son pruebas llevadas a cabo por los programadores sobre las unidades mínimas desarrolladas por ellos, estas unidades pueden clasificarse en **clases**, **métodos**, **propiedades**, **componentes**, etc. Estas unidades se prueban por separado y se hacen durante la implementación del software.

Para realizar estas pruebas es necesario establecer una serie de reglas que sirven como objetivos de estas pruebas:

- ✓ La prueba es un proceso de ejecución de un programa con la intención de descubrir errores.
- ✓ Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- ✓ Una prueba tiene éxito si descubre un error no detectado hasta entonces.
- ✓ El objetivo de diseñar pruebas se basa en que sistemáticamente se le muestren al programador diferentes tipos de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

Los “test de unidades” están orientados mayoritariamente a las pruebas de “caja blanca”; aunque para realizar uno de estos test es necesario probar el flujo de datos desde la interfaz del componente. Si los datos no entran correctamente, todas las demás pruebas carecen de sentido. Estas pruebas son aplicadas a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que estos funcionen como se espera.

Las “pruebas de caja blanca” se basan prácticamente en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de pruebas que examinan que estén correctos todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Estos casos de pruebas generan los caminos lógicos o posibles que debe recorrer el flujo de la ejecución, para afirmar que se esta ejecutando correctamente el procedimiento.

Una de las técnicas para ejecutar las pruebas de caja blanca es la del “Camino básico”, el resultado de esta técnica es una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática, define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Esta técnica fue usada anteriormente en este trabajo en el capítulo 2, epígrafe 2.2.1 “Análisis de complejidad del algoritmo”.

Otra forma de probar el código es mediante las pruebas de “caja negra”. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

Una de las técnicas mas usadas dentro de las pruebas de caja negra es la “Partición de Equivalencia” esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software. Es también una de las más efectivas, pues permite examinar los valores válidos y no válidos de las entradas existentes en el software.

Mediante esta técnica se descubren de forma inmediata una serie de errores que de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico, además se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así el número de clases de prueba que hay que desarrollar.
Para definir las clases de equivalencia es necesario tener en cuenta un conjunto de reglas:

- ✓ Si una condición de entrada especifica un rango, entonces se confeccionan una clase de equivalencia válida y 2 no válidas.
- ✓ Si una condición de entrada especifica la cantidad de valores, identificar una clase de equivalencia válida y dos no válidas.

- ✓ Si una condición de entrada especifica un conjunto de valores de entrada y existen razones para creer que el programa trata en forma diferente a cada uno de ellos, identificar una clase válida para cada uno de ellos y una clase no válida.
- ✓ Si una condición de entrada especifica una situación de tipo "debe ser", identificar una clase válida y una no válida.
- ✓ Si existe una razón para creer que el programa no trata de forma idéntica ciertos elementos pertenecientes a una clase, dividirla en clases de equivalencia menores.

Luego de tener las clases válidas y no válidas definidas, se procede a definir los casos de pruebas, pero para ello antes se debe haber asignado un identificador único a cada clase de equivalencia. Luego entonces se pueden definir los casos teniendo en cuenta lo siguiente:

- ✓ Escribir un nuevo caso que cubra tantas clases de equivalencias válidas no cubiertas como sea posible hasta que todas las clases de equivalencias hayan sido cubiertas por casos de prueba.
- ✓ Escribir un nuevo caso de prueba que cubra una y sólo una clase de equivalencia no válida hasta que todas las clases de equivalencias no válidas hayan sido cubiertas por casos de pruebas.

Con la aplicación de esa técnica se obtiene un conjunto de pruebas que reduce el número de casos de pruebas y dice algo sobre la presencia o ausencia de errores. A menudo se plantea que las pruebas a los software nunca terminan, simplemente se transfiere del desarrollador al cliente. Cada vez que el cliente usa el programa está llevando a cabo una prueba. Aplicando el diseño de casos de pruebas al software en cuestión se puede conseguir una prueba más completa y descubrir y corregir el mayor número de errores antes de que comiencen las "pruebas del cliente".

3.2 Aplicación de pruebas de caja blanca.

Para realizar este tipo de prueba es necesario primeramente realizar los procesos que se describieron en el capítulo 2, epígrafe 2.2.1 "Análisis de complejidad del algoritmo" para determinar los valores de complejidad ciclomática del algoritmo al cual se le va a aplicar la prueba. La función a la que se le va a aplicar la prueba es la función de borrar un negocio de la UDDI.

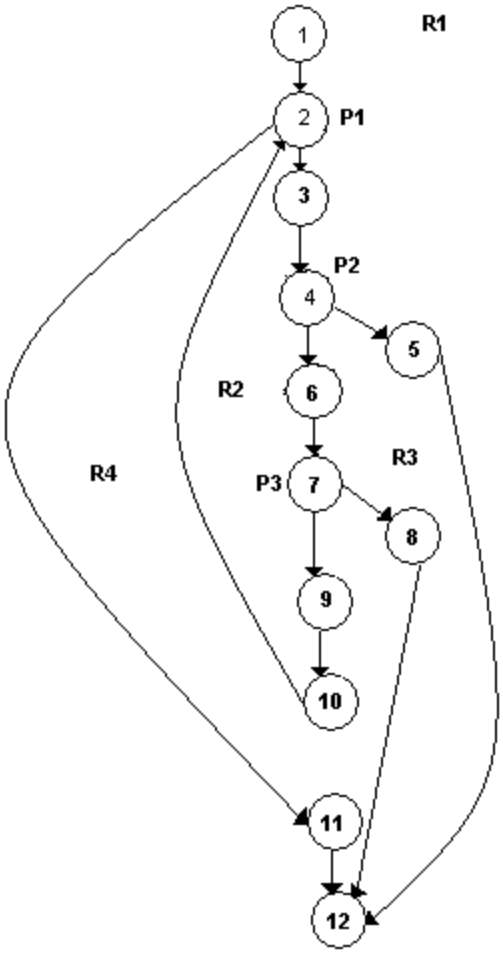
A continuación se enumeran las sentencias de código del procedimiento:

```
public function execute($regObject)
{
                $request = $regObject; 1
                $generic = $request->getGeneric(); 1
```

```
$authInfo = $request->getAuthInfo(); 1
$businessKeyVector = $request->getBusinessKeyVector(); 1


$dataStore = new JDBCDataStore(); 1

        $dataStore->beginTrans(); 1


        $publisher = $this->getPublisher($authInfo,$dataStore); 1
        $publisherID = $publisher->getPublisherID(); 1


        for ($i=0; $i<sizeof($businessKeyVector); $i++) 2
        {

                $businessKey = $businessKeyVector[$i]; 3


                if ((!$dataStore->isValidBusinessKey($businessKey))) 4
                {
                        throw new InvalidKeyPassedException("delete_business:
                        "."businessKey=".$businessKey); 5
                } 6



                if (!$dataStore->isBusinessPublisher($businessKey, $publisherID)) 7
                {
                        throw new UserMismatchException("delete_business: "."userID=".$publisherID.",
                        "."businessKey=".$businessKey); 8
                } 9

                 $businessKey = $businessKeyVector[$i]; 10
                $dataStore->deleteBusiness($businessKey);10

        } 11



        $dataStore->commit(); 12


        $dataStore->release(); 12


$result = new Result(); 12
$result = new Result($result->E_SUCCESS); 12
$result->setErrCode($result->lookupErrCode($result->E_SUCCESS)); 12
$dispRpt = new DispositionReport(); 12
$dispRpt->setGeneric($generic); 12
$dispRpt->addResult($result); 12

return $dispRpt; 12
} 13
```

Seguidamente se construye el grafo de flujo asociado al código anterior.

Figura XXIII. Grafo de flujo del algoritmo.



Seguidamente a la construcción del grafo de flujo se procede a efectuar el cálculo de la complejidad ciclomática del código, el cálculo es necesario efectuarlo mediante tres vías o fórmulas que para concluir que el cálculo fue correcto es necesario que por las tres vías el resultado sea el mismo, las fórmulas para calcular son las siguientes:

1. V (G) = R número de regiones del grafo de flujo.

2. V (G) = A - N + 2

Donde: **A** es el número de aristas del grafo y **N** es el número de nodos.

3. V (G) = *P* + 1

Donde: ***P*** es el número de nodos predicado (nodo del cual salen varias aristas) contenidos en el grafo ***G***.

Total de regiones R = 4

Aristas A = 15

Nodos N = 13

Nodos P = 3

V (G) = **R**

V (G) = **4**

V (G) = *A - N + 2*

V (G) = 15 - *13* + *2*

V (G) = 4

V (G) = *P* + 1

V (G) = 3 +*1*

V (G) = 4

Realizado el cálculo por las 3 vías necesarias se llega a la conclusión que el algoritmo presentado anteriormente tiene un complejidad ciclomática de 4 que da la visión de que existen a lo sumo 4 caminos lógicos por donde recorrer el algoritmo. Este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo. En estas representaciones se subrayan los elementos de cada camino que los hacen independientes de los demás.

Camino básico #1:

1 – 2 – 11 – 12– 13.

Camino básico #2:

1 – 2 – 3 – 4– 5 – 13.

Camino básico #3:

1 – 2 – 3 – 4– 6 – 7 – 8– 13.

Camino básico #4:
1 – 2 – 3 – 4– 6 – 7 – 9– 10 – 2– 11 – 12– 13.

Después de haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para este procedimiento, se debe realizar al menos un caso de prueba por cada camino básico, es necesario aclarar que la entrada de los datos al procedimiento tratado provienen de una clase controladora, en la cual el objeto DeleteBusiness, ya que este procedimiento se usa para eliminar business.

Para realizar los casos de pruebas es necesario cumplir con las siguientes exigencias:
Descripción: Se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no se entre algún dato erróneo,
Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.
Entrada: Se muestran los parámetros que entran al procedimiento
Resultados Esperados: Se expone resultado que se espera que devuelva el procedimiento.

Caso de prueba para el camino básico # 1.
Descripción: Los datos de entrada cumplirán con los siguientes requisitos: el parámetro recibido debe ser un objeto de tipo DeleteBusiness.
Condición de ejecución: el atributo businessKeyVector del objeto DeleteBusiness, estará vacio
Entrada: DeleteBusiness $request;
$vectorkey = $request->getBusinessKeyVector(); *($vectorkey es el atributo del objeto DeleteBusiness que estará vacio, longitud igual a 0)*
sizeof($vectorkey) =0 (*longitud del arreglo*)
Resultados esperados: Se espera que lance un error de tipo E_invalidKeyPassed.

Caso de prueba para el camino básico # 2.
Descripción: Los datos de entrada cumplirán con los mismos requisitos que el caso de prueba 1.
Condición de ejecución: el atributo businessKeyVector del objeto DeleteBusiness contendrá businessKey no válido o sea que no ha sido otorgado a nadie y por tanto no se encuentra en la base de datos.
Entrada: DeleteBusiness $request;
$vectorkey = $request->getBusinessKeyVector (); *($vectorkey es el atributo del objeto DeleteBusiness)*

$vectorkey [$pos] = 5252544 (*el valor del businessKey en $pos es no válido*).

Resultados esperados: Se espera que lance un error de tipo E_invalidKeyPassed.

Caso de prueba para el camino básico # 3.

Descripción: Los datos de entrada cumplirán con los mismos requisitos que el caso de prueba 1.

Condición de ejecución: el publicador identificado por el atributo $authInfo del objeto DeleteBusiness no le corresponderá el businessKey contenido en la posición $pos del atributo businessKeyVector del objeto DeleteBusiness.

Entrada: DeleteBusiness $request;

$vectorkey = $request->getBusinessKeyVector (); *($vectorkey es el atributo del objeto DeleteBusiness)*

$authInfo = $request-> getAuthInfo (); (authInfo perteneciente al objeto DeleteBusiness).

$publisherID (pertenece al publicador identificado con el authInfo anterior).

$authInfo = ‘ authToken:c5b2e1b6-7ea7-6298-eddf-eda2a8833157’

$publisherID = adminUDDI

$vectorkey [$pos] = dsd125e1b6-6kj8-e365-85421ljkh69’ (el valor del businessKey en $pos no pertenece al $publisherID = adminUDDI).

Resultados esperados: Se espera que lance un error de tipo UserMismatchException.

Caso de prueba para el camino básico # 4.

Descripción: Los datos de entrada cumplirán con los mismos requisitos que el caso de prueba 1.

Condición de ejecución: el publicador identificado por el atributo $authInfo del objeto DeleteBusiness le corresponderá el businessKey contenido en la posición $pos del atributo businessKeyVector del objeto DeleteBusiness, el atributo businessKeyVector del objeto DeleteBusiness contendrá un businessKey válido, el atributo businessKeyVector del objeto DeleteBusiness no estará vacio.

Entrada: DeleteBusiness $request;

$vectorkey = $request->getBusinessKeyVector (); *($vectorkey es el atributo del objeto DeleteBusiness)*

$authInfo = $request-> getAuthInfo (); (authInfo perteneciente al objeto DeleteBusiness).

$publisherID (pertenece al publicador identificado con el authInfo anterior).

$authInfo = ‘authToken: c5b2e1b6-7ea7-6298-eddf-eda2a8833157’

$publisherID = adminUDDI

$vectorkey [$pos] = d67ab2f5-91a4-90ed-53bb-b737008cdcd7’ (el valor del businessKey en $pos pertenece al $publisherID = adminUDDI).

Resultados esperados: Se espera que lance un mensaje vacio indicando que se elimino el Business correctamente.

Al comparar los resultados esperados con los resultados finales que se obtuvieron en los cuatro casos de prueba, se puede decir que fueron los mismos, por lo que se puede afirmar que el procedimiento que se tuvo en cuenta se encuentra en perfectas condiciones y el código no presenta ningún error.

En este capítulo se abordaron los diferentes métodos de pruebas que existen para determinar la calidad del software, que son los métodos de caja blanca y caja negra. Solo se hizo el análisis del método de caja blanca debido a que el software no cuenta con una aplicación visual. Las pruebas al software constituyen una herramienta más para que tenga la calidad requerida antes de ser entregado al cliente, para que el cliente quede satisfecho con el trabajo realizado y evitar trabajar doble.

Conclusiones

Como resultado de este trabajo se puede decir que se le dio cumplimiento al objetivo propuesto: se logró desarrollar la implementación de servicios para la publicación y validación de información en UDDI de acuerdo a los estándares y especificaciones internacionales.

Se puede decir además que se le dio cumplimiento a las principales tareas previstas:

- ✓ Se pueden calificar de satisfactorios los estudios realizados acerca del estado del arte de las APIs en UDDI en las diferentes versiones de UDDI existentes.
- ✓ Se logró un mayor entendimiento de las APIs UDDI planteadas en la especificación UDDI3.02 y se escogieron las APIs que se incorporarán a la UDDI que se esta desarrollando.
- ✓ La comparación de las APIs UDDI entre las diferentes versiones de UDDI existentes y la determinación de los cambios en las distintas versiones posibilitó que se implementaran las APIs de la forma más precisa y respetando los cambios.
- ✓ Fue muy provechoso el estudio del proceso de funcionamiento de cada servicio API en un registro UDDI porque facilitó una mejor visión en el momento de la implementación.
- ✓ Se logró aterrizar mejor lo que se quería implementar con el estudio de las APIs: UDDI Publication API, UDDI Custody and Ownership Transfer API, UDDI Value Set Caching API y UDDI Value Set Validation API en jUDDI.

Recomendaciones

Se recomienda para futuras iteraciones incluir:

- ✓ Implementación de la API UDDI Custody and Ownership Transfer API.

Referencias Bibliográficas

[1] Wikimedia Foundation, I. (s.f.). *Paradigma de programación*. Recuperado el 21 de 01 de 2008, de http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n

[2] Anónimo. (s.f.). *UNIVERSIDAD TECNOLOGICA, NACIONAL REGIONAL TUCUMAN - EPARTAMENTO DE SISTEMAS.* Recuperado el 8 de 02 de 2008, de http://www.frt.utn.edu.ar/sistemas/paradigmas/page22.html

[3] Desconocido. (s.f.). *Tesis de Ingeniería Informática.* Recuperado el 2 de 03 de 2008, de http://www.fi.uba.ar/laboratorios/lsi/yolis-tesisingenieriainformatica.pdf

[4] Rey, E. M. (s.f.). Recuperado el 27 de 02 de 2008, de Departamento de Computación, Universidad de Coruña, España: http://quegrande.org/apuntes/POO/teoria/T1-Introduccion.pdf

[5] S.A, M. (s.f.). *Paradigma Orientado Objetos.* Recuperado el 15 de 02 de 2008, de http://www.monografias.com/trabajos14/paradigma/paradigma_orientado_objetos

[6] Fundation, W. (s.f.). *Arquitectura Orientada a los Servicios*. Recuperado el 2008 de 01 de 16, de http://es.wikipedia.org/wiki/Arquitectura Orientada a los Servicios

[7] Wikimedia Foundation, I. (s.f.). *Wikipedia*. Recuperado el 13 de 01 de 2008, de http://es.wikipedia.org/wiki

[8] Consortium (W3C), W. W. (2005). Guía Breve de Tecnologías XML. Retrieved 10 14, 2007, from http://www.w3c.es/divulgacion/guiasbreves/TecnologiasXML

[9] 2002-2004, C. ©. (5 de 10 de 2002-2004). *OASIS UDDI Spec TC.* Recuperado el 08 de 01 de 2008, de http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[10] Msaffirio. (s.f.). Recuperado el 06 de 03 de 2008, de Qué son los Servicios web: http://msaffirio.wordpress.com

[11] Hernández, C. G. (2006). *Módulo Alojamiento del Sistema Automatizado para la Gestión de Información de la Misión Milagro.* Universidad de las Ciencias Informáticas, Ciudad de la Habana.

[12] Cantero, J. (2002-2004). *Un vistazo a PHP5.*

[13] Wikimedia Feijoó, E. T. (2004). *PHP5 El Lenguaje Para Los Profesionales De La Web.* Madrid: EDICIONES ANA YA Wikimedia MULTIMEDIA (GRUPO ANAYA, S.A.), 2004.

Bibliografía

Consortium (W3C), W. W. (2005). Guía Breve de Servicios Web. Retrieved 10 14, 2007, from http://www.w3c.es/Divulgacion/Guiasbreves/ServiciosWeb

Consortium (W3C), W. W. (2007, 06 26). Web Services Description Language (WSDL) Version 2.0 Part 1:

Core Language. Retrieved 10 17, 2007, from http://www.w3.org/TR/wsdl20/

Granado, L. M. (24, 10 2007). Wordpress. Retrieved 11 05, 2007, from http://sabuesoweb.wordpress.com/2007/10/24/manual-imprescindible-de-php-5/

Group, T. S. (2002, 07 19). Retrieved 12 03, 2007, from http://www.uddi.org/pubs/the_evolution_of_uddi_20020719.pdf

IDG COMMUNICATIONS, S. (2007, 02 16). Retrieved 10 11, 2007, from SOA aporta a la Administración Pública un nuevo paradigma para mejorar la interoperabilidad entre sus aplicaciones. Su implantación requiere la aceptación de estándares y normas comunes: http://www.idg.es/computerworld/articulo

Martínez, P. E. (2007). EXTENSIBILIDAD DE UDDI. Universidad de Sevilla: Departamento de Lenguajes y Sistemas Informáticos.

Nacional, F. R.-U. (2004-2005). Paradigmas de Programación. Retrieved 03 04, 2008, from http://paradigmas.firebirds.com.ar/

Pacheco, A. (2008, 06 26). Paradigmas de Programación. Retrieved 03 02, 2008, from http://expo.itch.edu.mx/view.php?f=prog_10

Rolando, H. L. (2002). EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA. Ciudad de La Habana: EDUNIV, Editorial Universitaria, 2002. ISBN: 959-16-0343-6.

Web, D. (n.d.). Manual PHP. Retrieved 10 23, 2007, from http://www.desarrolloweb.com/php/

Anexos


Paradigmas Programación
Especifican la programación como un conjunto de secuencias computacionales que se ejecutan paso a paso
No existe una descripción paso a paso de cómo llegar a la solución, sino que el programa se construye estableciendo hechos, reglas, restricciones, ecuaciones, etc. Que debe cumplir el conjunto de valores que constituyen la solución.
Se provee al sistema de soluciones a problemas similares y se deja que generalice una solución a partir de estos ejemplos.
Paradigmas de programación
Paradigma operacional
Paradigma declarativo
Paradigma demostrativo
Paradigma imperativo
Paradigma de orientación a objetos
Paradigma funcional
Paradigma lógico
Paradigma transformacional
Paradigma relacional
Paradigma de inducción
Paradigma de redes de neuronas
Paradigma genético
Haskell, Erlang
Prolog
Sistemas basados en reglas
SQL

Glosario de Términos

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

Web Services: Aplicación que realiza un cometido y que puede formar parte de otros servicios para formar un servicio más completo. La comunicación hacia y desde el Web Service se realiza con XML. Permite una llamada a una funcionalidad localizada en un servidor remoto.

OASIS: acrónimo de (Organization for the Advancement of Structured Information Standards) es un consorcio internacional sin fines de lucro que orienta el desarrollo, la convergencia y la adopción de los estándares e-business.

XML: son las siglas en inglés de eXtensible Markup Language (lenguaje de marcado ampliable o extensible) desarrollado por el World Wide Web Consortium (W3C). No es más que un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados. XML tiene otras aplicaciones entre las que destaca su uso como estándar para el intercambio de datos entre diversas aplicaciones o software con lenguajes privados como en el caso de SOAP.

SOAP: (siglas de Simple Object Access Protocol) es un protocolo estándar creado por Microsoft, IBM y otros, está actualmente bajo el auspicio de la W3C que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos utilizados en los servicios Web.