UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 5: Entornos Virtuales y Automática

ANÁLISIS Y DISEÑO ORIENTADO A SERVICIO DEL MÓDULO DE REPORTES DEL SCADA

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

AUTOR: Dayrien Corrales Díaz.

TUTOR: Ing. Amado Espinosa Hidalgo.

Ciudad de la Habana, julio 2008.

"Año 50 del Triunfo de la Revolución"

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 año _____.

_____

Firma del Autor
(Dayrien Corrales Díaz)

_____

Firma del Tutor
(Ing. Amado Espinosa Hidalgo)

Datos de contacto

Ing. Amado Espinosa Hidalgo

Graduado de Ingeniero Informático en el 2004 y profesor instructor con cuatro años de experiencia docente en la Universidad de las Ciencias Informáticas (UCI) y cinco en el desarrollo de software. Correo electrónico: aespinosa@uci.cu

Dedicatoria

A mi madre, mi abuela, a mi hermanita,

a Frank y a Lazy;

quisiera que mi abuelo hubiese podido estar.

A Yamila.

...a ustedes por significar tanto en mi vida.

Agradecimientos

A mis compañeros de grupo y del proyecto.

A los amigos que aportaron ideas.

A quienes me aclararon tantas dudas.

A las profesoras como Lida, Mayra.

A Yorji.

A Amado, mi tutor.

A mi madre por su dedicación y preocupación.

A mi tío Frank.

A mi padre.

A todos los que me apoyaron.

Resumen

Un sistema de Supervisión Control y Adquisición de Datos (SCADA) se encarga de monitorear los eventos que tienen lugar en estaciones remotas. En la Universidad de las Ciencias Informáticas el Proyecto SCADA (también conocido por Guardián de la Alternativa Bolivariana para las Américas) desarrolla un sistema de este tipo.

El trabajo se descompuso en varios módulos, entre ellos el módulo de Reportes, el cual permite la generación de informes. La arquitectura implantada en la primera versión no brinda la posibilidad de tener acceso remoto a las funcionalidades creadas para la generación de los informes utilizados por clientes u otros módulos del sistema.

Este trabajo de diploma propone un modelado del módulo aplicando la Arquitectura Orientada a Servicios (SOA) con el objetivo de obtener una nueva versión de la herramienta. El diseño final sirve de preámbulo para el desarrollo del software con orientación a servicios. Está centrado en el análisis y diseño de los servicios que se pueden definir en el módulo, así como la descripción de la interacción que ocurra en cada uno de ellos. Los servicios propuestos no están asociados con la visualización de los resultados que brinda; estas funciones se ejecutan sin que el usuario lo note, abstrayéndolo de sus funcionalidades.

PALABRAS CLAVES

Arquitectura Orientada a Servicios, SOA, Servicios y Mensajes, Orientación a Servicios, Integración de Sistemas, Sistemas Distribuidos.

Abstract

Title: Service – Oriented Analysis and Design for Report Module of SCADA system.

Author: Dayrien Corrales Diaz.

Tutor: Engineer Amado Espinosa Hidalgo.

A system for Supervision Control and Data Acquisition (SCADA) is used to monitoring the events occur in remote stations. In University of Informatics Sciences the SCADA Project develops a system like this.

Work was decomposed in several modules, among them Report Module have to give the possibility to generate information about SCADA state in useful formats. The architecture previously defined in the first version can not give a remote access to the functionalities to create these reports.

This career work proposes a software modeling using Service – Oriented Architecture (SOA) looking for develops new software. The final design is directed to the service-oriented analysis and design for services defined in the module.

Keywords:

SOA, Service Oriented Architecture, Service Oriented, Systems Integration, Service and Messaging, Distributed Systems.

Tablas y Figuras

Contenido

INTRODUCCIÓN

Los sistemas de supervisión, control y adquisición de datos son conocidos por el acrónimo SCADA. Estos permiten el control y la monitorización de los procesos que ocurren en estaciones remotas. El constante desarrollo de software empresarial para estos tiene como concepto fundamental el uso de herramientas que permitan la generación de informes a partir de datos preestablecidos. Los desarrolladores tienen en cuenta la importancia de estos sistemas generadores que proporcionen una rápida creación de informes y una rápida modificación y mantenimiento.

En el año 2006, en el marco de las relaciones de convenio entre Cuba y Venezuela, se crea en la universidad el Proyecto SCADA (también nombrado Guardián de la Alternativa Bolivariana para las Américas) con alumnos y profesores de la Facultad 5, en el cual se está desarrollando un SCADA (valga la redundancia). El mismo será utilizado en el control automático de los procesos industriales que se efectúan en la empresa PDVSA de la hermana República Bolivariana de Venezuela.

Durante la construcción del mismo se creó un módulo para generar los reportes, a través de los cuales se puede obtener el estado de los tanques y otras variables comprendidas en el proceso industrial. Esta herramienta en su primera versión se diseñó e implementó con una arquitectura en capas usando la orientación a objetos.

Se desea que tanto dentro o fuera de las fronteras del SCADA se pueda acceder a las funcionalidades que permiten la generación de los reportes. Para esto la arquitectura actual no brinda la posibilidad de definir componentes lógicos que encapsulen dichas operaciones y proporcionaran el acceso remoto a las mismas. Con ello se persigue que el modulo de reportes tenga independencia de la plataforma, el lenguaje y la tecnología. La versión actual no satisface estos

aspectos por lo que surge la necesidad de enfocar el desarrollo del módulo hacia otra arquitectura. Analizando lo anterior se formula la siguiente interrogante:

¿Cómo modelar el generador de reportes del SCADA para que sus funcionalidades de crear informes puedan ser accesibles a través de la red desde cualquier lugar en cualquier momento? Este constituye el problema científico a resolver.

Desde el año 2000 en la industria de software, la Arquitectura Orientada a Servicios (SOA) ha tomado gran aceptación por los desarrolladores. Esta arquitectura ofrece interfaces expuestas en la red que encapsulan las funcionalidades del sistema. Usando estándares se puede conocer lo que hacen las mismas y como interactuar con ellas. Así se logra obtener un producto independiente de la plataforma y el lenguaje de desarrollo; y neutral a la tecnología que se utilice tanto para la creación del mismo como para posterior invocación.

El objetivo general trazado consiste en rediseñar el módulo de reportes aplicando la arquitectura orientada a servicios para definir componentes lógicos que expongan las funcionalidades de esta herramienta y así lograr que sean accesibles mediante la red.

Podríamos ver el ejemplo en que un usuario desee obtener un reporte en su celular, pues cuenta con un software que le permite hacer la solicitud para ello. Mediante el uso de los estándares estas operaciones se pueden efectuar, pero la arquitectura actual que soporta el sistema generador no da la posibilidad de que esto se lleve a cabo.

El presente trabajo de diploma propone el modelado de la herramienta tomando SOA como la arquitectura del mismo. Por tanto el objeto de estudio en que se basa este trabajo es el Análisis y Diseño Orientado a Servicios, teniendo como campo de acción el Análisis y Diseño Orientado a Servicios del Módulo de Reportes.

Para realizar una investigación a fondo de esta arquitectura se tuvo como objetivos:

- Determinar conceptos y principios de la arquitectura.
- Determinar los estándares utilizados por la arquitectura que sean aplicables a la herramienta.
- Identificar los elementos necesarios que componen la arquitectura para desarrollar una aplicación.
- Proponer una tecnología para el desarrollo.
- Realizar un levantamiento de requisitos.
- Adoptar una estrategia para el análisis y diseño.
- Desarrollar un prototipo funcional.

En la investigación se adoptaron los métodos teóricos: Histórico-Lógico al realizarse un estudio del surgimiento y análisis de SOA, abordando además en su evolución. Analítico-Sintético al definirse los conceptos y principios de la arquitectura. Inductivo-Deductivo cuando se selecciona la tecnología, después de un estudio de las existentes, a utilizar en el flujo de trabajo de Implementación con vista a obtener un producto.

La estructura del presente trabajo de diploma consta de 4 capítulos. El próximo capítulo contiene toda la fundamentación teórica del tema en cuestión. El capítulo 2 expone una serie de tecnologías que soportan SOA, se define la metodología usada, se propone el entorno en que se desarrollará la solución y los elementos reutilizables.

El resto del trabajo está centrado en los flujos de trabajo: Modelo del Negocio, Captura de Requisitos y Análisis y Diseño proporcionando una entrada al posterior desarrollo de la solución.

CAPÍTULO 1: "Fundamentación Teórica."

Introducción.

En la industria de software se han ido elaborando diferentes metodologías que abogan por la rapidez en la respuesta de los sistemas de información. Los sistemas distribuidos juegan un papel protagónico en este aspecto, gracias a cómo el trabajo se reparte entre varias estaciones de trabajo, las cuales exponen dichas funcionalidades en servicios a través de la red, que al integrarse en una infraestructura funciona como un todo.

Una vía para lograr esto es diseñar con la Arquitectura Orientada a Servicios el sistema a desarrollar, pues permite traducir las funcionalidades en servicios que, distribuidos en una red, proporcionan una estrategia de organización para las tareas del negocio expuestas en elementos de las Tecnologías de Información (TI), proporcionando una mayor y mejor producción.

En este capítulo se brinda brevemente información del surgimiento y evolución de SOA, se exponen conceptos que ayudan a entender cual es el funcionamiento de un sistema desarrollado con orientación a servicios, sus componentes, utilidad y efecto que ha tenido en el mundo y en la industria de software. También se realizan comparaciones que se realizaron entre la arquitectura actual y la propuesta para la solución.

1.1 Surgimiento y evolución de SOA.

La Arquitectura Orientada a Servicios se ha puesto de moda en el desarrollo de sistemas de estos últimos años, aunque no es un concepto nuevo en la industria de software. Se estima que a mediados de la década de 1980, cuando tuvo surgimiento las llamadas a procedimientos remotos y los primeros sistemas distribuidos, comenzó a hacer sus primeras apariciones en el mercado. Durante esta década y los años de 1990 el uso de SOA se vio limitado en los sistemas y

no es hasta el año 2000 que alcanza su más alto galardón, convirtiéndose en el estilo de arquitectura más importante si de desarrollo de aplicaciones se trata.

La Consultora Internacional Gartner se refirió a SOA por primera vez en el año 1996, dando una pequeña predicción de que en la actualidad se convertiría en la vía de solución del desarrollo de sistemas empresariales. A partir de la década pasada los arquitectos dejaron de pensar en su aplicación como algo aislado de las demás aplicaciones que existían en diferentes plataformas y lenguajes. Se lanzaron entonces a diseñar sistemas interoperables entre sí, dando lugar al paradigma de aplicaciones en capas, estrechamente relacionado con Programación Orientada a Objetos (OOP).

Hasta aquí las aplicaciones distribuidas tenían una debilidad lamentable: no permitían las comunicaciones con aplicaciones de otras plataformas. Para finales de los 90, Microsoft había desarrollado una tecnología que permitía la comunicación entre las capas de las aplicaciones, lo que era muy factible en el mundo del gigante azul, pero a la hora de interactuar con otro sistema operativo, por ejemplo GNU/Linux, había que lidiar con protocolos de bajo nivel modificando interfaces COM, haciendo pasar las llamadas a través de firewalls; todo esto más otra cantidad de complicaciones. Esto dio nacimiento a los Servicios Web (WS).

Los WS son aplicaciones que corren ocultas al usuario y permiten la comunicación basada en estándares entre cualquier plataforma. Los estándares son utilizados en el transporte, codificación e intercambio de información. SOAP, XML y HTTP son los más utilizados, exponiendo en lenguaje WSDL las funcionalidades que brinda el servicio. La aceptación en la gran comunidad informática ha hecho evolucionar los WS dando pie a la Arquitectura Orientada a Servicios.

Los principios de esta arquitectura no son tan actuales, pero es ahora en la actualidad que existen los estándares que permiten la interoperabilidad necesaria para llevarse a cabo en la práctica. Las anteriores tecnologías solo

brindaban un soporte parcial por parte del mercado. Debido a esto se limitaba la interoperabilidad y todo lo demás, porque sin estándares no hay interoperabilidad, ni reutilización, ni SOA. [**1**]

El pasado año Gartner predijo que para el 2008 SOA ha de proporcionar la base de más del 80% de todos los proyectos de desarrollo y permitirá a las empresas aumentar la reutilización del código en más del 100% de los casos, ya que las empresas utilizarán esta arquitectura como un "principio rector" al crear aplicaciones de misión crítica y procesos, gracias a la gama de ventajas que brinda persiguiéndose o no la integración. [**8**]

1.2 ¿Qué es SOA?

Este concepto puede ser visto con muchas definiciones que en ocasiones se centran en los aspectos técnicos de la arquitectura y a veces incluyen las características del negocio sumadas a cómo el software está actualmente implementado. No obstante todas las definiciones pueden ser resumidas y vistas en una sola.

SOA es una arquitectura de software multicapa que se caracteriza por tener las funcionalidades del sistema que desarrolla descompuestas en servicios agrupados por interfaces, que, de forma independiente y autónoma, se publican para que a través la red puedan ser accesibles. [**45**]

Al lograr esta descomposición de los procesos de negocio en servicios se consigue separar el QUÉ del CÓMO. SOA se inicia con la definición de una interfaz y construye la aplicación en topología de interfaces. Debido a esto todo aquel que solicite un servicio no sabrá cómo es que se lleva a cabo el mismo. Los componentes están dados por mensajes, operaciones, servicio y procesos.

Con SOA es posible la creación de procesos de negocios a partir de la perspectiva de las Tecnologías de Información. Esta arquitectura no está atada a

una tecnología específica. La implementación de un sistema puede llevarse a cabo utilizando un amplio rango de tecnologías, dentro de las que se pueden incluir: REST, RPC, DCOM, CORBA. [**2**]

Puede servir como soporte para la integración de diferentes sistemas de integración. Aunque es muy visto en los Servicios Web vale aclarar que no es lo mismo. Dicha arquitectura más bien es una abstracción del éxito de los WS.

Se puede implementar con cualquier tipo de tecnología cliente – servidor, pero no es para todo tipo de aplicaciones. Tampoco es una tecnología, ni una herramienta. En el mercado existen muchos productos que han sido desarrollados con orientación a servicios, pero sería un error pensar que es una arquitectura de rápida implementación. [**3**]

Se tiende a confundir el concepto de Software como Servicio (SaaS) con SOA por la aparente semejanza que se aprecia entre ellos. Evidentemente no son lo mismo y eso se puede comprobar en el funcionamiento que realiza cada uno.

1.2.1 ¿Qué es un Software como Servicios (SaaS)?

Concepto muy ligado a SOA, SaaS no es otra cosa que el software que se pone en explotación con la modalidad de servicio del sistema operativo que se usa. Esta aplicación puede ser accedida desde la red. Este concepto suele asociarse con los proveedores de servicios de aplicación.

Se crearon para compartir datos con un solo destinatario. Tiene una capacidad limitada de compartir datos y procesos con otras aplicaciones por lo que tendían a ser escasamente atractivas en comparación con las aplicaciones instaladas localmente. En la actualidad estas aplicaciones pretenden aprovechar las ventajas de la centralización y ofrecer una experiencia con funcionalidades avanzadas que hagan competencia ventajosa frente a las aplicaciones de instalación local. [**4**]

Esto tiene su origen en ASP donde se usara para el almacenamiento centralizado de paquetes de software, los cuales luego de esto se instalaban en una serie de clientes que pagaban por ello. [**5**]

Tanto SOA como SaaS se ven en otro concepto: Composición de Aplicaciones, también mal llamado Mashups. La Composición de Aplicaciones expresa una perspectiva de la ISW que define la construcción de una aplicación mediante la combinación de múltiples funciones unidas en una aplicación nueva. En muchas ocasiones la composición de aplicaciones requiere de orquestación de las funciones o servicios encontrados en la red [**6**]. Mientras, Mashups no es más que un sitio web que combina datos de más de una fuente de información, logrando integrar estas como una solo herramienta. [**7**]

Debido a estos conceptos las personas muchas veces tienden a comparar los conceptos de composición de aplicaciones con los Mashups. Lo cierto es que ambos usan la reutilización de funciones existente. La finalidad de estos términos no difiere mucho en cuanto a la integración e interoperabilidad de las funciones reutilizables que se exponen en la red como servicios.

1.3 ¿Qué es un sistema SOA?

Este concepto es visto como la composición de todos los servicios que operan conjuntamente como un todo para un fin específico. Los sistemas basados en la orientación a servicios son vistos como sistemas distribuidos. Los cuales pueden contener servicios de otros sistemas que se encuentran en la red.

SOA surge con el objetivo de construir sistemas de bajo acoplamiento con el uso de tecnologías como XML Schema, SOAP, WSDL, HTTP, UDDI, entre otros. Los arquitectos que diseñen sistemas basados en dicha arquitectura son los encargados de disminuir el acoplamiento de las funcionalidades del sistema, dicho en otras palabras: eliminar las dependencias que puedan existir entre ellas en el runtime. [**9**]

Para el control de la comunicación entre los servicios que componen un sistema SOA, se utilizan tecnologías que se basan en la orquestación de servicios. Esto consiste en obtener una conexión total de los servicios para crear procesos de negocios de alto nivel, lo cual se logra con la unión de las entidades del sistema.

1.4 ¿Por qué usar SOA?

SOA permite un mejor alineamiento de las Tecnologías de Información con las necesidades de negocio, permitiendo a empleados, clientes y socios comerciales responder de forma más rápida y adaptarse adecuadamente a las presiones del mercado.

El desarrollo de aplicaciones es rápido y flexible, pero esto no es instantáneo. Es necesario un proceso lento para ser capaz de asimilar los nuevos comportamientos que implica usar este nuevo paradigma para el desarrollo de aplicaciones. La utilización de todos los principios de Orientación a Servicio es algo complejo.

Otras de las ventajas que proporciona el uso de esta arquitectura son la independencia de plataforma, la escalabilidad, más agilidad de organización, mayor calidad en los servicios y la reducción de riesgos durante el desarrollo y soporte del software. En pocas palabras, permite la creación y cambio de servicios de forma incremental, evitando proyectos de larga duración y alto coste.

Se requiere menos comunicación y hay menos sobrecargas de recursos. Es más fácil de implementar y de dar mantenimiento. Existe menos complejidad cuando el consumidor o el proveedor deben ser cambiados. Solo se requiere acordar procesos o estados a nivel de negocios. Proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

Se puede interactuar con gran cantidad de tecnologías existentes. Los servicios se ven como una única entidad. En su implementación puede encontrarse cualquier nivel de complejidad. No importa que estén implementados en cualquier lenguaje siempre y cuando ellos puedan generar e interactuar con WSDL. [**10**]

1.5 Nociones sobre SOA.

Para lograr un buen diseño de un sistema SOA existen 3 pasos básicos. Primero: Debe existir una buena traducción de los requerimientos a los casos de uso. Segundo: Se debe definir cada proceso de negocio como servicio del sistema. Tercero: Definir una tecnología que emplee un orquestador de servicios o middleware para nuestro sistema en caso que se desee la integración. Esto es opcional para aplicaciones que exponen servicios sencillos y de fácil acceso.

Para el desarrollo de un sistema SOA existen varias metodologías para el diseño e implementación. Dada la independencia de los servicios que componen el sistema se utiliza un orquestador que dirige, controla, gestiona las operaciones de encontrar, conectar y ejecutar servicios. Para lograr esto se debe cumplir con los principios de orientación a servicios establecidos.

La metodología de modelado y diseño para aplicaciones SOA se conoce como análisis y diseño orientado a servicios. Esta arquitectura es tanto un marco de trabajo para el desarrollo de software como un marco de trabajo de implantación. Para que un proyecto de este tipo 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 estos sistemas requiere un compromiso con este modelo en términos de planificación, herramientas e infraestructura.

Cuando se habla de una arquitectura orientada a servicios, se está hablando de un juego de servicios residentes en Internet o en una Intranet, no

necesariamente usando servicios web. 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.

1.5.1 Servicios.

Se conoce como servicio las funcionalidades de un sistema encapsuladas en una interfaz que se expone en la red, conocidas también como procesos públicos (portType). Estos pueden ser tan simples como conocer la hora de la maquina en la que se ejecuta el servicio, o tan complejo que un mismo servicio tenga que estar compuesto por otros servicios. La comunicación entre ellos no es más que el intercambio de datos públicos, lo cual se conoce como mensajes (message). Estos se pueden clasificar según su funcionalidad de la siguiente forma:

- Servicios Controladores o de Comunicación, los encargados de coordinar el trabajo con el resto de los servicios a través de la mensajería que se utiliza en la comunicación entre ellos. No contienen el estado de las aplicaciones aunque muchas veces se configuran para concertar el trabajo con los sistemas que los usan. Estos servicios no manejan los datos que contienen los mensajes, solo usan parte de ellos para saber a donde direccionar los datos de los mensajes.

- Servicios de Negocio, representan una tarea muy puntual del negocio, formando parte de este. Por lo general emplea servicios de utilidad para dar cumplimiento a la tarea que representa. Estos servicios pueden obtenerse a partir de los casos de uso.

- Servicios de Utilidad, ejercen una tarea altamente reutilizable en el sistema y forman parte de la implementación de un proceso de negocio. Son vistos formando parte de servicios que pueden ser usados en una o más aplicaciones, encapsulando funcionalidades como cálculos complejos o acceso a datos, ya sean centrados a datos o a lógica.

Por lo tanto una aplicación SOA se puede ver en una gráfica como la interacción de los servicios rigiendo el proceso de negocio en un sistema.

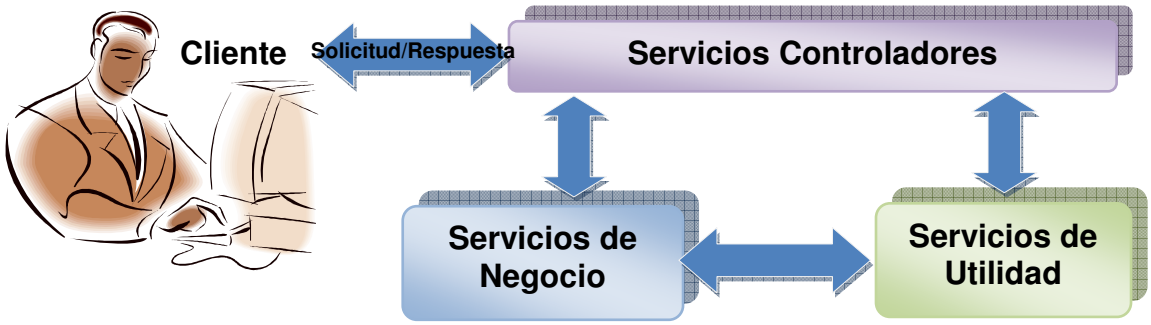


Figura: 1.1 Relación entre servicios.

Los servicios están compuestos por una interfaz en la cual se exponen las operaciones que forman parte de este servicio. La implementación de estas funciones contiene solo lógica, no se almacenan datos en las mismas, aunque por detrás de estas puede existir OOP. Independientemente de esto los servicios deben cumplir con los Principios de Orientación a Servicios establecidos.

1.5.2 Mensajería

El término consiste en el intercambio de datos entre servicios. El mensaje es el mecanismo de comunicación el cual viaja por un canal de comunicación establecido.

El intercambio de datos entre los servicios se establece una vez definidos y configurados los canales de comunicación (Endpoint) entre los mismos. Estos se pueden definir usando los estándares de comunicación actuales o mediante código. Endpoints se estructuran de la forma ABC (Address, Binding, Contract).

Address corresponde a la dirección física del servicio en la red ya que estos son accedidos a través de la URI especificada en el contrato. Binding contiene el protocolo de transporte, codificación y requerimiento de seguridad para la

comunicación entre los servicios. Contact, contrato del servicio constituido por las funciones que ejecuta, datos que se intercambian, ubicación de la descripción, localización y nombre del mismo. Se refiere a que hace el servicio y como interactuar con él.

Los desarrolladores son los encargados de modelar el contrato. Esto se puede hacer mediante el lenguaje de programación o directamente en un XML. Para ello primero se definen los mensajes a través de clases y/o entidades del sistema. Luego se crean las funciones que operarán con los mensajes, las cuales se modelan de forma independiente de su exposición. Por último se agrupan estas operaciones en interfaces, las cuales se exponen en la red.

La mensajería se define a partir de los patrones ya establecidos. Estos describen cual va a ser el estado de los servicios que interactúan en el intercambio y el ciclo de vida de los mensajes que intervienen en el mismo. Estos patrones de mensajería son:

1. Petición – Respuesta (Request – Response). El cliente envía una solicitud al servidor y queda bloqueado en espera de una respuesta.

2. Solo Petición (Only Way). El cliente envía una solicitud al servidor y no necesita respuesta, por lo que no queda bloqueado.

3. Request – Callback. Este es un tipo de mensajería request - response asíncrona. El cliente envía una solicitud al servidor y no queda bloqueado en espera de una respuesta.

4. Publicación - Suscripción (Publish – Subscribe). El cliente envía una solicitud al servidor y queda bloqueado en espera de una respuesta porque esta petición que no necesita una respuesta.

5. Mensajes de Error. Mensajes de error producidos entre servicios ya sea porque el contrato está mal diseñado o el binding no es correcto.

6. EDA (Event-Driven Architecture). Mensajería establecida por eventos.

Aquí se genera el mensaje y se envían por eventos.

En un sistema SOA puede que no se vean todos los patrones de mensajería pero sí deben cumplirse con todos los principios de orientación a servicios en cada uno de los servicios definidos en el sistema.

1.5.3 Entidades de un SOA.

Un sistema bajo esta arquitectura está compuesto por varias entidades. Estas entidades se configuran de forma conjunta para poder encontrar, enlazar y ejecutar servicios.

1. Consumidores de servicios.
2. Proveedores de servicios.
3. Registros de servicios.
4. Contratos de servicios.
5. Proxys de servicios.

Consumidor de servicio es todo aquel, servicio, aplicación, módulo de software, que requiere de un servicio determinado. Esta entidad luego de encontrar la dirección del servicio se enlaza sobre un transporte y ejecuta la función requerida del servicio, enviando una solicitud con el formato que se adecúa al contrato del servicio.

Proveedor de servicio son las entidades que expuestas y accesibles mediante la red aceptan y ejecuta solicitudes desde los consumidores. Este puede ser un mainframe, un componen u otro sistema se software que ejecute la solicitud recibida. El contrato se publica en el registro para que pueda ser accedido por los consumidores para que configuren las solicitudes.

Registro de servicio es un directorio basado en red que publica los servicios disponibles. Entidad que acepta y almacena los contratos de servicios contenidos en proveedores y los expone a los consumidores interesados.

Contrato de servicio es la especificación de la forma en que los consumidores deben interactuar con los proveedores de servicios. Esta especificación define el formato que se debe elaborar la solicitud y respuesta de un servicio. El contrato puede requerir en un momento precondiciones y poscondiciones para realizar solicitudes.

Proxy de servicio es el API que se encarga de encontrar el contrato y la referencia del proveedor del servicio en el registro. El consumidor al efectuar una solicitud llama a una función del API en el proxy. Se formatea el mensaje de la solicitud y ejecuta la misma en representación del consumidor. [**11**]

Un proxy de servicios puede ser clasificado como repositorio de servicios (UDDI), pues la base del funcionamiento es la misma. Los UDDI son vistos en los WS donde brindan información de contrato, localización, descubrimiento dinámico y acuerdos a nivel de servicios.

1.6 Orientación a Servicios.

Al desarrollar una aplicación debemos entender que hace factible la definición de sus componentes para obtener un producto verdaderamente orientado a servicios. Software dirigido a las empresas puede traducir su lógica en servicios. SOA introduce nuevos conceptos a través de los cuales se representan, modelan y comparten los procesos de negocios. Con ello los servicios entablarán una mayor forma de abstracción entre las capas de negocio y aplicación, encapsulando la lógica tanto de negocio como de aplicación. [**12**]

El diseño e implementación de los servicios se lleva a cabo teniendo en cuenta los Principios de Orientación a Servicios, los cuales definen en teoría el

funcionamiento y comportamiento de los mismos. Esta teoría no especifica la plataforma o lenguaje de programación para el desarrollo del sistema, solo los aspectos relevantes como medio de apoyo para lograr la separación de preocupaciones.

Principios de Orientación a Servicios:

1. Los servicios deben ser reutilizables.
2. Los servicios deben proporcionar un contrato formal.
3. Los servicios deben permitir la composición.
4. Los servicios resumen lógica subyacente.
5. Los servicios deben tener bajo acoplamiento.
6. Los servicios deben ser autónomos.
7. Los servicios no deben tener estado.
8. Los servicios deben poder ser descubiertos.

Para logra un sistema SOA los servicios deben estar diseñados bajo estos principios. Lograr esto es un poco difícil por lo que en ocasiones es necesario contactar con expertos en el área. [**13**]

1.6.1 SOA vs OOP

La relación que existen entre orientación a servicios y la orientación a objetos no representan una competencia entre ellas. OOP se utiliza comúnmente en la construcción de sistemas SOA. Sin embargo la metodología de ambas difiere. La orientación a servicios complementa la orientación a objetos introduciéndole mejoras.

SOA: Los sistemas son diseñados con un bajo nivel de acoplamiento que facilita la implementación de cambios, y estos pueden ser desarrollados en cualquier lenguaje corriendo incluso en diferentes plataformas.

OOP: Los sistemas de objetos distribuidos están fuertemente acoplados y modificar una parte de un sistema de este tipo implica romper alguna otra parte, salvo que esa otra parte sea también modificada al mismo tiempo.

SOA: Con este nuevo paradigma se encapsula la funcionalidad en servicios independientes, lo cual conlleva a una orquestación de la ejecución de los servicios brindando una alta interoperabilidad entre los mismos. El alcance de una unidad de procesamiento lógico puede llegar a variar significativamente.

OOP: Este paradigma de la programación se caracteriza por los conceptos de Encapsulamiento, Polimorfismo y Herencia. El encapsulamiento de la información se realiza en clases interdependientes que garantiza una alta reusabilidad del código para crear otras aplicaciones a partir de las ya creadas. El alcance de una unidad lógica tiende a ser menos amplio. La orientación a objetos practica la reutilización a través de clases.

SOA: Existen pocas interacciones de granularidad gruesa. Cada interacción es la misma, o sea, es un paso en un proceso de negocio lo cual no produce procesos o estados de más bajo nivel.

OOP: Existen muchas interacciones de granularidad fina. Cada interacción es diferente, por lo que hacen falta muchas interacciones para completar un proceso de negocio, lo que provoca que haya muchos procesos o estados intermedios. [**15**]

1.6.2 Capas de servicios.

Las interfaces que exponen las funcionalidades que efectúan los servicios, se ubica entre las capas de negocio y aplicación. Aquí es donde tiene mayor

prevalencia las características de SOA. Para lograr el bajo acoplamiento que se quiere es necesario separar las capas físicas de los servicios. Estas capas logran la abstracción para la solución que encierra cada funcionalidad. [**16**]

Capas de abstracción con las que podemos definir SOA:

1. Capa de servicios de utilidad.
2. Capa de servicios de negocio.
3. Capa de orquestación de servicios o servicios controladores.

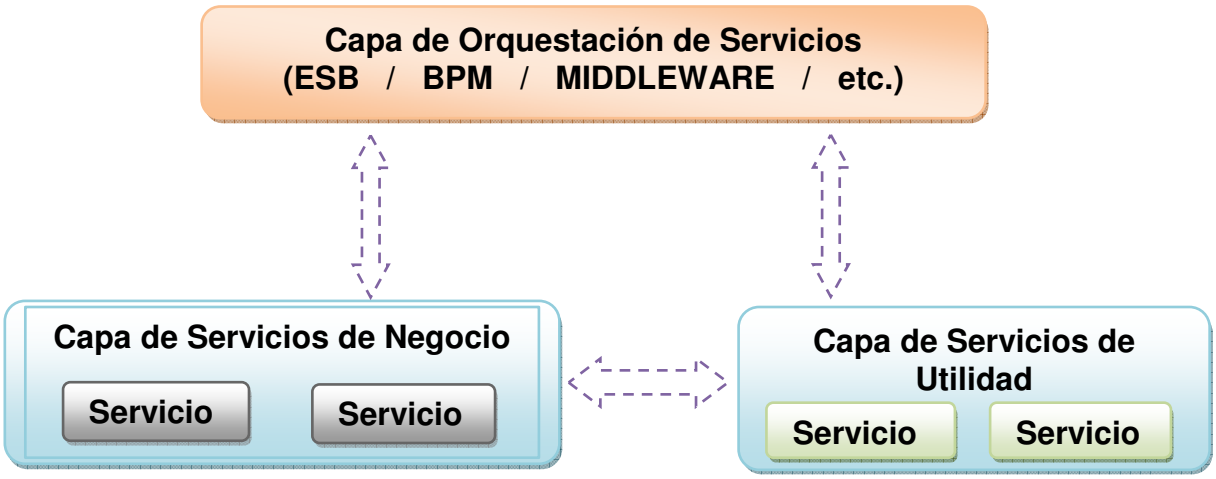


Figura: 1.2 Capas de servicios de un sistema SOA.

Capa de Servicios de Aplicación.

E la responsable de representar funcionalmente la tecnología y lógica de la aplicación. En ella se coleccionarán los servicios que sus funciones tengan un intercambio directo con la capa de aplicaciones. Provee funciones reusables que estén estrechamente relacionadas con el procesamiento de datos de la capa de aplicaciones, para aprovechar los recursos disponibles dentro de una plataforma. [**16**]

Capa de Servicios de Negocio.

Esta es la capa mediante la cual se expresa la lógica del negocio a través de la orientación a servicios. Los servicios contenidos estarán dirigidos a cumplir una tarea del negocio. Representa la lógica del negocio de la forma más pura posible, implementando en servicios el modelo de negocio. [**17**]

Capa de Orquestación de Servicios.

Constituida por el o los servicios de procesos de negocio, más conocidos por servicios controladores o de comunicación. Se gestiona en detalles la interacción de las peticiones que se realizan sobre un servicio, se comprueban los requerimientos de las mismas, asegurándose con las operaciones que el servicio ejecuta. Se provee un conjunto específico de funciones, independientemente de las reglas de negocio y la lógica que se ejecuta en un proceso de negocio. [**18**]

1.7 Análisis y Diseño Orientado a Servicios.

El modelado de la orientación a servicios requiere de actividades y artefactos adicionales que no están presentes en el tradicional Análisis y Diseño orientado a objetos. Existen 3 elementos claves a tener en cuenta cuando se va a modelar un SOA: Servicios, Flujo entre ellos y su Composición. También debe ocurrir un cambio de paradigma con el fin de identificar las dos funciones claves: Proveedor de servicios y Consumidor de servicios. [**28**]

El ciclo de vida de un proyecto SOA está compuesto por una serie de pasos que deben completarse para la construcción de servicios de una determinada solución orientada a servicios, valga la redundancia. Las fases básicas que se

pueden apreciar para ello, abarcan todo el desarrollo de esto elementos: Análisis, Diseño, Desarrollo, Pruebas, Despliegue y Administración orientada a servicios. [**38**]

Este trabajo solo se hace uso de las dos primeras fases, lo cual se deja plasmado visualmente en diagramas obtenidos tras modelar servicios.

1.7.1 Descripción general

- **Análisis orientado a servicios.**

Se denomina análisis orientado a servicios al proceso necesario para representar la automatización de los procesos de negocio a través de la orientación a servicios.

El objetivo de esto es definir un conjunto preliminar de candidatos a la orientación a servicios. Se definen las fronteras preliminares para que no se solapen otros servicios ya previstos o existentes. Se identifica la lógica encapsulada con potencial reusable. Hay que asegurarse que la lógica encapsulada es apropiada para la utilidad que se quiere de ella.

- **Diseño orientado a servicios.**

El diseño orientado a servicio es a través del cual se concreta el diseño físico de los servicios, estos se reúnen en una composición abstracta que implementa un proceso o una tarea puntual del negocio.

Con el diseño se determina el conjunto básico de extensiones arquitectural. Se establecen límites en la arquitectura. Se identifican los diseños estándares necesarios y el potencial de composición de los servicios. Se definen los diseños

de interfaces abstractas de servicios. Se evalúa además el soporte para los principios de orientación a servicios.

- **Estrategias a valorar.**

El trabajo con estas etapas se organiza a través de una estrategia, se conocen tres: Top-Down, Bottom-Up y Agile (meet in the middle). El uso de las mismas esta estrechamente relacionado con las prioridades de la organización para establecer el correcto equilibrio entre el suministro a largo plazo de objetivos y el cumplimiento de necesidades a corto plazo. [**35**]

Top – Down.

Esta estrategia logra a gran medida un análisis en primer lugar. Para ello se requiere no solo procesos de negocio a convertir en orientación a servicios, sino también promueve la organización del modelo de negocio. Además está estrechamente vinculada a la lógica del negocio existente.

El proceso de esta estrategia recorre desde la definición de las ontologías relevantes al alineamiento de los procesos de negocio, para luego obtener el análisis de los servicios y continuar hasta el despliegue de los mismos. [**37**] **Ver Anexo 1.**

Bottom – Up.

La integración es el principal impulsor de esta estrategia. Este enfoque fomenta fundamentalmente la creación de servicios como un medio para cumplir los requisitos centrados en la aplicación. La necesidad de aprovechar las ventajas de las comunicaciones abiertas de SOAP puede cumplirse simplemente añadiendo servicios que recubran sistemas heredados.

Se parte del modelo de los servicios de aplicación para luego obtener un diseño de estos, dando paso al desarrollo y prueba de los mismos y en último lugar el despliegue. [**38**] **Ver Anexo 2.**

Agile (meet-in-the-middle).

Para esta estrategia el objetivo sigue siendo obtener un equilibrio aceptable entre la incorporación de los principios de diseño de la orientación a servicios en el entorno de análisis del negocio sin necesidad de esperar una integración anterior de las tecnologías de los WS. Esto es posible definiendo nuevos procesos que permitan un análisis a un nivel de negocio paralelamente al diseño y desarrollo de servicios.

Esta estrategia se alinea con el estado actual del negocio a partir del análisis que se realiza en el mismo. Durante la elaboración del proyecto se vela por la ocurrencia de cualquier cambio que ocurra en los requerimientos del cliente. En caso que esto ocurra se gestiona el rediseño de los servicios y con ello una reestructuración de las demás fases de desarrollo. [**39**] **Ver Anexo 3.**

Conclusiones.

La arquitectura orientada a servicios no nace de una propuesta académica, es más bien el resultado de toda la experiencia acumulada en el área de computación distribuida y desarrollo de sistemas interoperables y heterogéneos. Existe un impulso por desarrollar sistemas SOA o simplemente migrar a este. Se cuenta con suficiente tecnología para lograrlo. La mayoría de las entidades que usan IT abogan por la integración y esta arquitectura supone una estrategia para ello.

La orientación a servicios está muy ligada a la orientación a objetos. Sabiendo cuales son las diferencias que existen entre ellas se puede lograr que trabajen juntas. No se pueden ver como dos metodologías que compiten porque entre ellas hay más semejanza de la que aparentan.

Introducción.

En este capitulo se hace referencia a las tecnologías que soportan SOA. Se selecciona la candidata para la construcción de la solución. Se describen el entorno de de trabajo y los componentes reutilizables así como los estándares que se utilizarán. Se definen el soporte para la modelación y la base para el desarrollo.

2.1 Tecnologías que soportan SOA.

SOA es una arquitectura que se puede implementar con múltiples tecnologías, gracias a los estándares universalmente soportados existentes en la actualidad. Se ha adoptado a XML como el lenguaje para codificar los mensajes. El mismo permite definir las estructuras de datos según el contrato que se exponga de una interfaz. Es fácil de comprender y de manipular por los desarrolladores de software. Es un poco pesado para enviar por la red y lento para ser parseado facilita el intercambio de la información a través de mensajes.

XML es ideal para esta arquitectura porque logra la estandarización de la interoperabilidad y la composición de la información, tomando los datos, los trata como componentes y se convierte en un contenedor de ellos. Habilita definiciones, transmisiones y validaciones de los mismos.

Las tecnologías que soportan SOA están centradas en la integración de datos, aplicaciones, componentes. Esto se logra de forma sincrónica o asíncrona. Los mecanismos como sockets, RPC, RMI, DCOM, colas, tópicos, son elementos a tener en cuenta por los paradigmas que lleven a cabo este proceso. [**21**]

2.1.1 Middleware

Es un software que posibilita la el intercambio entre aplicaciones distribuidas. Este software corre sin que el usuario lo note. Logra una abstracción de la complejidad de las redes y de los sistemas operativos entre los que ocurre el intercambio. Por lo general están orientados a mensajes, visto por como el servidor almacena una cola de mensajes de forma asíncrona. En la actualidad existe una nueva tecnología llamada XML Middleware que afecta todas las categorías de Middleware por la tendencia hacia la estandarización para la interoperabilidad, favorecida por el uso del dicho estándar. [**22**]

2.1.2 CORBA

Arquitectura estándar para sistemas de objetos distribuidos. Los servicios que un objeto provee son dados por su interfaz. Tienen definido un contrato a través de la interfaz que contiene los objetos. De igual manera las solicitudes pueden estar escritas en un lenguaje de programación diferente a la implementación de los objetos. En CORBA se añade la interoperabilidad como una meta en la especificación, para ello define un conjunto de servicios distribuidos que soportan además la integración. [**23**]

Es una de las tecnologías sólidas y mejor diseñadas. Con el apoyo que tiene mundialmente ha llegado a la integración con XML, para la mensajería entre los objetos. [**25**]

2.1.3 SOAP

Primer protocolo de comunicación bajo HTTP mediante XML. Ha sido aceptado por la mayoría de los productores de software del mundo. Entre sus ventajas se puede encontrar que no está asociado a ningún lenguaje, incluso no especifica

ninguna API, lo cual se deja al lenguaje de programación. SOAP no se encuentra fuertemente asociado ningún protocolo de transporte.

No está atado a ninguna infraestructura de objeto distribuido. Se aprovechan los estándares existentes en la industria, como por ejemplo: XML para codificar los mensajes, HTTP o SMTP para el transporte. Permite la interoperabilidad entre múltiples entornos. Utiliza WSDL para describir la interfaz pública de los WS, requisitos del protocolo y formatos de los mensajes. Usa UDDI, como registro en el catálogo, para recibir los mensajes y dar paso al WSDL. [**26**]

2.1.4 RPC.

Una llamada a procedimientos remotos (RPC) es la invocación de un programa que está en una computadora por otro programa que se encuentra en otra computadora. La petición de un cliente y la respuesta a esa petición constituye una transacción RPC. Sun Microsystems desarrolló esta tecnología y hoy día se ha adaptado para usar XML con protocolo de transporte HTTP, dándose a conocer como XML-RPC. La característica fundamental de esta tecnología es realizar infrecuentes transacciones relativamente pequeñas entre las llamadas.

Otros protocolos RPC basados en HTTP son: SOAP y CORBA, los más famosos. El más reciente en el mundo es REST-RPC. [**26**]

2.1.5 BPM.

La administración de procesos de negocio (BPM) es una recopilación de técnicas y herramientas de SW con el objetivo de modelar, gestionar y optimizar procesos de negocio. Persiguiendo mejorar la eficiencia a través de la gestión sistemática de los procesos de negocio. Las soluciones BPM facilitan la redefinición y automatización de los procesos de negocio con el acortamiento de la duración de los mismos y la simplificación de los errores. Para dar

cumplimiento al ciclo de vida de un BPM existen herramientas que dan el soporte necesario para cumplir con esto, estas herramientas son conocidas por BPMS (sistema de administración de procesos de negocio). [**26**]

2.1.6 ESB.

El bus de servicios empresariales trata a todas las aplicaciones que controla como un servicio. Es capaz de conectar y coordinar centenas de servicios mediante la combinación de WS, XML, gestión y transformación de datos. Esta tecnología se plasma en la red integrada por nodos desplegados en contenedores de servicios. Con ESB se practica una integración centrada en estándares al desplegar los contenedores en partes específicas de la red, conectándose a una topología de bus lógico mediante servidores de comunicación.

De cierta forma ESB combina los paradigmas SOA y EDA. Además implementa interfaces estandarizadas para proveer comunicación, conectividad, transporte, portabilidad y seguridad para con los servicios. [**26**]

2.1.7 REST.

Este término nacido en el año 2000 en una tesis doctoral sobre la Web, escrita por Roy Fielding, es el estilo arquitectónico para sistemas hipermedia distribuidos por la WWW. Describe cualquier interfaz bajo los estándares de XML y HTTP, excluyendo abstracciones de patrones de intercambio de mensajes. Opera mediante las funciones definidas por HTTP: GET, POST, PUT, DELETE; el conjunto de tipos de contenidos identificados a través de tipos MIME. REST no es un estándar.

Tanto los WS como SOAP han logrado revolucionar los sistemas distribuidos con el uso de los estándares y las tecnologías desarrolladas por W3C. REST

llega a mostrar una serie de limitaciones en el modelo WS-SOAP, que pudieran superarse si se adoptara un enfoque distinto. Como dicho modelo está basado en RPC, define un conjunto de componentes que usando REST bastaría definirlos en uno solo.

Con este estilo arquitectónico se pueden contemplar ventajas como: mejores tiempos de respuestas y disminución de carga en el servidor, facilidad de desarrollo de clientes. Mayor estabilidad frente a futuros cambios. Facilidad de desarrollo de clientes. [**26**]

2.2 UML como soporte de la modelación de la solución propuesta.

Este lenguaje se ha convertido en un estándar para la modelación de las fases de los proyectos informáticos: desde el análisis con casos de uso, diseño con diagrama de clases, objetos, etc., hasta la implementación y configuración de los diagramas de despliegue. Con el Lenguaje Unificado de Modelado (UML) se logra que una abstracción de la realidad se plasme en una notación gráfica conocida como modelado visual. Sirve además para el modelado de sistemas complejos, tanto de software como de arquitectura de hardware donde se ejecuten.

También hay que tener en cuenta que UML es independiente del lenguaje de implementación en que se desarrolle el producto. Muestra mayor rigor en la especificación de los elementos. Permite verificar y validar lo que ha sido modelado para automatizar procesos. Se puede generar código a partir de lo modelado y viceversa.

Mediante este lenguaje se puede establecer una comunicación. Se visualiza de forma gráfica el sistema. Se especifican cuales son las características antes de su construcción. Con este lenguaje dichos sistemas modelados se pueden construir. Los elementos gráficos realizados sirven de documentación del

sistema desarrollado para quien desee conocer el sistema sin necesidad de ver el código de la implementación o el sistema en ejecución.

2.3 RUP como base en el desarrollo de la solución.

El Proceso Unificado de Desarrollo (RUP) es una propuesta metodológica por los creadores de UML. Está basado en componentes. Usa el Lenguaje Unificado de Modelado para la representación gráfica de los esquemas de un sistema software.

El proceso de desarrollo para la construcción de la solución está definido por los tres aspectos que rigen el proceso unificado de desarrollo: dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental. Con esta metodología podemos abarcar todo el ciclo de vida: desde el levantamiento de requisitos hasta dar el acabado del sistema, proporcionando una visión coherente y completa de la construcción del producto.

La solución propuesta está basada en casos de uso, a partir de lo cuales se pueden crear una serie de modelos de diseños y de implementación para ayudar al entendimiento de los desarrolladores. IBM propone una vía para construir sistemas SOA mediante esta metodología. En las diferentes fases que se aborda la construcción de la solución se pueden apreciar iteraciones características de RUP, las cuales se depuran, fortalecen y mejoran después de cada hito.

No se puede olvidar que RUP unifica los mejores elementos de metodologías anteriores. Está preparado para desarrollar grandes y complejos proyectos. Además utiliza UML como representación visual. Estos aspectos hacen que desde el primer momento en que se piensa crear un producto software o mejorar uno existente, se piense en este proceso de desarrollo.

2.4 IBM Rational Software Architect.

Es un entorno de modelación y desarrollo que aprovecha el Lenguaje Unificado de Modelado versión 2.0 (UML v2.0) para diseñar arquitecturas de software soportadas por C++, Java 2 Enterprise Edition (J2EE) y aplicaciones Web. Aprovecha al máximo la plataforma abierta Eclipse 3.2. Ofrece una gama completa de herramientas de desarrollo de software para una variedad de tecnologías de implementación.

Las muchas características de modelado y edición visual muy poderosas de la herramienta están diseñadas para mejorar la productividad, elevar el control arquitectónico y facilitar la experiencia del diseño al código para J2EE, C++, la Arquitectura Orientada a Servicios (SOA) y los servicios Web.

Rational Software Architect (RSA) ayuda a migrar con más facilidad de una herramienta de desarrollo a otra. Permite agregar características a medida que cambien las necesidades, lo que hace del producto una selección consumible para las necesidades de diseño y de construcción. Además es único en su habilidad de dar soporte de modelado a otros dominios tales como WSDL y XSD (XML Schema Definition), permitiéndole crear diagramas que combinan los elementos de UML con estos últimos. [**44**]

2.5 Entorno de desarrollo de la solución.

La Comunidad de Software Libre ha tomado la filosofía de compartir el conocimiento para la solución de problemas. Así el desarrollo de software se basa en la reutilización. Esta filosofía de trabajo trae consigo que para crear un producto, o tan solo instalar uno, existan ciertas dependencias soportadas en librerías o bibliotecas. En la construcción de la solución se hace uso de varias, necesarias para el desarrollo.

Los demás módulos del SCADA siguen una línea de desarrollo usando lenguajes de programación de alto nivel bajo la metodología orientada a objetos.

Se desarrolla en el lenguaje C++, usando el Entorno de Desarrollo Integrado (IDE) Eclipse y C/C++ Development Tool (CDT), plug-in que permite la implementación en C/C++.

Eclipse es una herramienta abierta y se coloca entre los IDE mas utilizados en el desarrollo de software. Permite la refactorización, actualización / instalación de código gracias a Update Manager. Es neutral a la plataforma y al lenguaje de desarrollo. Emplea diversos módulos de los cuales obtiene su funcionalidad por lo que puede extenderse usando otros lenguajes de programación, editores de texto y aplicaciones de red.

Otra herramienta a tener en cuenta es la consola Unix contenida en toda distribución GNU/Linux. Con esta se generan los skeleton o stubs tanto para el cliente como para el servidor a través de una herramienta proporcionada por gSOAP-2.7.10. Además la compilación por parte de los desarrolladores puede realizarse de forma óptima.

2.5.1 Componentes reutilizables

Los servicios de la solución propuesta se desarrollan como WS publicados en un servidor Apache. Para la creación de los mismos contamos con la librería gSOAP, versión 2.7.10, la cual nos posibilita el trabajo en el lenguaje elegido para su confección. Es libre y de fácil entendimiento. Se encarga de establecer el protocolo SOAP para la comunicación entre las interfaces que expone nuestro sistema.

Con esta librería se obtiene una API que oculta detalles irrelevantes al usuario. Es compatible con SOAP 1.1 y SOAP 1.2 aunque aun se encuentra en fase de desarrollo. También brinda un mecanismo de para la transmisión y optimización mensajes entre los servicios. MTOM es relativamente un nuevo formato para la enviar archivos a través de WS.

Antes resultaba un problema enviar archivos como mensaje sin antes serializarlo. Este problema queda resuelto con esta librería ya que la misma se encarga de dicho proceso.

Otra librería a usar es la GSL (GNU Scientist Library) para llevar a cabo las operaciones estadísticas que se deseen efectuar en la gestión de reportes. Contiene múltiples herramientas con utilidad altamente científica en cálculos y operaciones.

Bajo estas condiciones se adoptan los mismos estándares utilizados para el desarrollo de WS. La comunicación entre los servicios se llevará a cabo a través de HTTP. Los mensajes que existan en el intercambio de datos serán contenidos en XML, mediante un protocolo SOAP. Para el caso de las imágenes que se crearán como SVG.

Conclusiones.

En este capítulo nos centramos en las tecnologías que pudieran sernos útiles para el desarrollo de la solución. De ellas escogimos una y desglosamos los elementos que con carácter reutilizable se emplearán para obtener un producto altamente confiable e interoperable.

Introducción.

En este capítulo se describen las características del sistema partiendo de un modelo de dominio. Se reflejan los primeros artefactos obtenidos en el modelo de negocio y captura de requisitos, los cuales muestran una visión de lo que se quiere desarrollar. De identifican los servicios candidatos para conformar la solución.

3.1 Entorno donde trabajará el sistema.

El SCADA en el que operará el sistema está constituido por varios módulos que trabajan conjuntamente para el control de los procesos industriales. Los reportes que se deseen generar deberán gestionarse en herramientas capaces de comunicarse con los componentes encargados de producirlos.

Esta acción la realiza un servicio Web ubicado en un servidor. Los sistemas que se comunican con él deben ser capaces de crear mensajes XML con el formato establecido para el intercambio.

3.1.1 Eventos principales.

Para obtener un reporte, primeramente se debe enviar un mensaje con la configuración establecida para a comunicación con el servicio que se encarga de generar el mismo. Dicha solicitud contiene los datos que se quieren reflejar en el informe: configuración de conexión a la BD, consultas a evaluar, etc. Además se debe reflejar el formato en que se desea el documento. Una vez completado el informe se precede a establecer el formato en el que se desea obtener el mismo.

Una vez conformado el reporte se devuelve este al dueño de la solicitud.

3.1.2 Diagrama de clases del Modelo de Dominio

Como punto de partida de este negocio se conoce que no tiene fronteras bien definidas, por tanto se establece un modelo de dominio en el que se describen las distintas entidades que participan, así como las relaciones entre ellas. En el siguiente diagrama de clases de objeto muestra una vista estructural de ello.

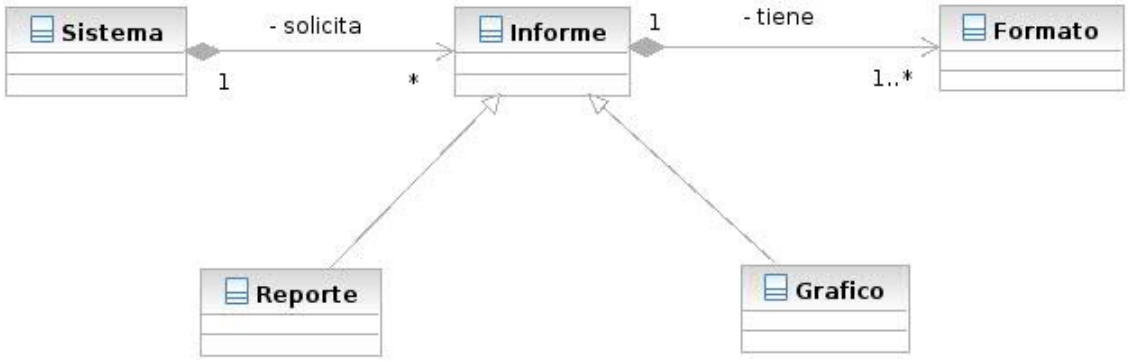


Figura: 3.1 Diagrama de clases del Modelo de Dominio.

A continuación se explican los conceptos tratados en el modelo de dominio:

Sistema: Define todas las aplicaciones que deseen comunicarse con nuestros servicios con el fin de obtener un informe. Estas deben ser capaces de interactuar con WS mediante WSDL.

Informe: Identifica los informes que se generen a partir de una solicitud recibida de un sistema. Los mensajes recibidos tendrán la característica definida en el contrato de la interfaz del servicio.

Formato: Especifica que a partir de la configuración del mensaje recibido por el servicio se le establece un formato conque el documento se debe enviar como respuesta a la solicitud realizada por Sistema.

Reporte: Es una especialización de informe que esta compuesto por información textual (variables establecidas en el SCADA y valores de cada una).

Gráfico: Es una especialización de informe que contiene información visual en forma gráfico de barras, líneas o pastel. Refleja valores de variables y parámetros establecidos en el SCADA.

3.2 Requisitos del Sistema.

Los requisitos rigen el desarrollo de los servicios necesarios a construir para dar respuesta al objetivo de este trabajo. Aquí se definen cuales son las funcionalidades del sistema. Lo que se expone seguidamente se tuvo en cuenta a la hora de identificar los servicios que se definieron para componer este módulo. Los mismos representan funcionalidades potenciales y posibles de optimizar.

3.2.1 Requisitos Funcionales.

A continuación se enumeran las capacidades o condiciones que el sistema debe cumplir.

1. RF1. Generar reporte.

- RF1.1 Generar reporte con formato de documento PDF.
- RF1.2 Generar reporte con formato de documento CSV.
- RF1.3 Generar reporte con formato de documento HTML.

2. RF3. Generar gráficos estadísticos

- RF3.1 Generar gráficos de líneas.
- RF3.2 Generar gráficos de barras.
- RF3.3 Generar gráficos de pastel.

3.2.2 Requisitos no Funcionales.

Cualidades que debe tener el producto para dar cumplimiento a los requisitos funcionales.

1. RNF1. Confiabilidad.

- RNF1.1 Permitir que los servicios se mantengan activos y accesibles durante el tiempo que el SCADA esté en funcionamiento.

2. RNF2. Mantenimiento.

- RNF2.1 Permitir mantenimiento independiente a los demás subsistemas del SCADA.

3. RFN3. Rendimiento.

- RNF3.1 Permitir tiempo de respuesta de las operaciones de petición de reportes con sensación de inmediatez.

4. RNF4. Interoperabilidad.

- RNF4.1 Permitir que con los servicios que brinda puedan intercambiar datos con otros subsistemas dentro y fuera del SCADA con permiso para ello.

5. RNF5. Portabilidad.

- RNF5.1 Permitir que los servicios puedan ser desarrollados en diversos lenguajes y plataformas.
- RNF5.2 Permitir que los servicios que brinde sean accedidos desde diferentes plataformas y por diferentes aplicaciones creadas en cualquier lenguaje.

6. RNF6. Software Libre

- RNF6.1 El sistema será software libre por lo tanto todos los componentes que utilice deben ser libres.

3.3 Descripción del Sistema Propuesto.

El objetivo de nuestro sistema es que pueda ser accedido no solo desde las estaciones clientes sino también por otros módulos u otras entidades. Cualquiera que necesite realizar un proceso definido en nuestra aplicación, sin la explotación de muchos recursos, invoca al servicio publicado en un servidor Apache.

Los reportes se generarán directamente en un archivo con formato seleccionado por el usuario según su conveniencia. Estos archivos son creados por el servicio una vez hechas las consultas a la base de datos y antes de ser enviados se serializan en un archivo con formato XML.

Un componente muy importante a tener en cuenta de los reportes es los gráficos que se puedan generar a partir de datos seleccionados por el usuario. Antes del surgimiento de las imágenes SVG esto se tornaba un tanto difícil. Hoy en día el empleo que se les ha dado y la aceptación que tiene en las plataformas fundamentalmente en GNU/Linux ha hecho de este un elemento imprescindible para la transmisión de imágenes a través de la Web.

3.3.1 Descripción de los actores

Tabla1: Descripción de los Actores.

Actor del Sistema	Justificación
Sistema	Es la representación de todo sistema que desee interactuar con nuestros servicios. Dichos sistemas pueden ser módulos del SCADA u otra aplicación fuera de las fronteras del mismo.

3.3.2 Caso de Uso del Sistema

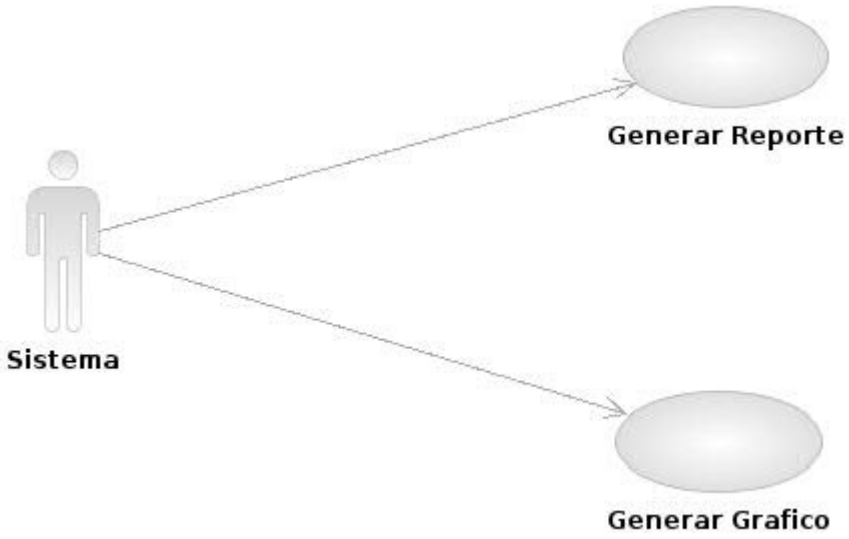


Figura: 3.2 Diagrama de CU del sistema.

3.3.3 Descripción Caso de Uso del sistema.

Tabla2: Descripción del Caso de uso Generar Reporte.

Nombre del Caso de Uso	**Generar Reporte**
Actores	Sistema (inicia)
Propósito	Permitir que los usuarios obtener información a través de informes o reportes con el formato deseado.
Resumen	El caso de uso inicia cuando Sistema envía una solicitud enviada al proveedor. El servicio genera el reporte y luego lo envía con el formato establecido al consumidor.

Referencias	RF1.
Precondiciones	El mensaje de solicitud debe estar correctamente configurado, según el contrato del servicio.
Poscondiciones	El sistema permite realizar la operación deseada, según el mensaje de solicitud.

Curso normal de los eventos	
Acciones del Actor	**Respuesta del Sistema**
1. El consumidor envía la solicitud al proveedor del servicio, con los datos de las consultas, variables, parámetros, y elementos en los que desea reflejar la información.	2. El servicio determina si la solicitud es válida. 3. Obtiene la configuración contenida en la solicitud. 4. Realiza las operaciones de consulta a la base de datos para obtener los datos a reflejar en el reporte. 5. Establece el formato deseado al reporte conformado con los datos de la consulta. 6. Serializa el documento para que pueda ser enviado al consumidor. 7. El servicio envía el informe con el formato obtenido como respuesta a la solicitud del consumidor.
8. El consumidor del servicio recibe el informe solicitado	
Curso alterno de eventos	

	2.1. En caso que la solicitud no sea correcta envía un mensaje de error al consumidor.
8.1 El consumidor del servicio recibe la notificación del error.	
Prioridad	Crítica.

Tabla3: Descripción del Caso de uso Generar Gráfico.

Nombre del Caso de Uso	**Generar Gráfico**
Actores	Sistema (inicia)
Propósito	Permitir a los usuarios obtener gráficos estadísticos a partir de datos seleccionados: parámetros a medir y valores de cada uno.
Resumen	El caso de uso inicia cuando Sistema envía una solicitud al proveedor del servicio. El servicio genera el grafico con formato de líneas, de barras o de pastel y lo envía al consumidor.
Referencias	RF3.
Precondiciones	El mensaje de solicitud debe estar correctamente configurado, según el contrato del servicio.
Poscondiciones	El sistema permite realizar la operación deseada, según el mensaje de solicitud.
Curso normal de los eventos	

Acciones del Actor	**Respuesta del Sistema**
1. El consumidor envía un mensaje de solicitud al proveedor del servicio con los datos para crear el gráfico: parámetros y valores a reflejar.	2. El servicio valida los datos de la solicitud. 3. Crea la plantilla de la imagen a enviar con los parámetros que se desean reflejar. 4. Establece los valores a visualizar en la grafica. 5. Calcula las coordenadas para los valores de los parámetros. 6. Pinta en la imagen los valores a reflejar. 7. Envía la imagen SVG al consumidor como respuesta a la solicitud recibida.
8. El consumidor del servicio recibe el informe solicitado	
Curso alternativo de los eventos	
	2.1 En caso que la solicitud no sea correcta envía un mensaje de error al consumidor.
8.1 El consumidor del servicio recibe la notificación del error.	
Prioridad	Crítica.

3.4 Servicios candidatos.

De lo analizado anteriormente se pueden identificar dos servicios candidatos. Estos estarán agrupados en la misma capa, la cual puede dar solución a las peticiones, dando cumplimiento con los requisitos definidos.

Para cada servicio se define el contrato que va a cumplir en la etapa de construcción del sistema. Para mayor entendimiento se definen mensajes, funciones y una breve descripción de lo que realiza el servicio, así como otros parámetros que se tienen en cuenta durante el desarrollo de los mismos.

3.4.1 Servicio Generar Reporte

Nombre del Servicio: Generar Reporte.
Versión: 1.0
Propietario: Proyecto SCADA
Tipo de servicio: Servicio de Negocio.
Requisito Funcional: RF1. Generar Reporte.
Capa**:** Servicio de Negocio.
Datos:

Mensaje de Solicitud: Datos configurados por el usuario.

Mensaje de Respuesta: Reporte obtenido con el formato deseado.

Operación:

Generar Reporte PDF.

Generar Reporte CSV.

Generar Reporte HTML.

Invocación: http://url_del_servidor/cgi-bin/generateReport.cgi
Descripción: http://url_del_servidor/services/generateReport.wsdl
Documentación:

Servicio definido por la función encargada de generar reportes a partir de la configuración que el usuario le da reporte. Efectúa una consulta a la BD. Luego utiliza otro servicio para establecer en el informe el formato

deseado por el usuario, el cual envía como respuesta a la solicitud del consumidor del servicio.

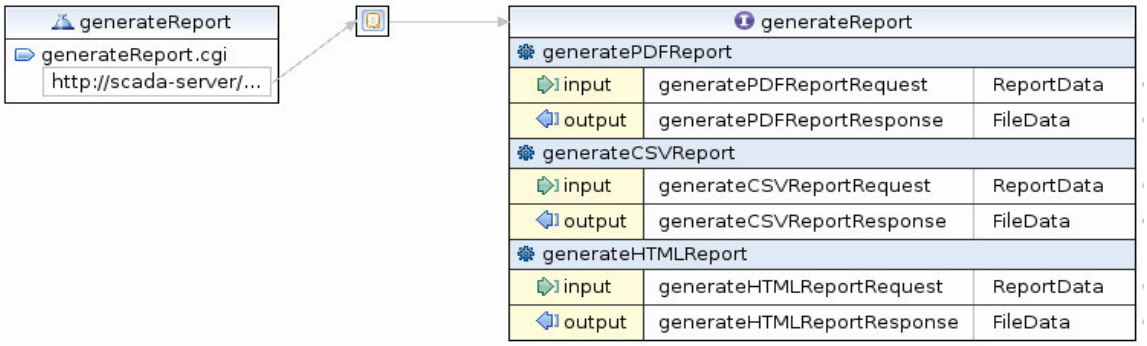


Figura: 3.3 Representación del contrato para el servicios Generar Reporte.

3.4.2 Servicio para Generar Gráficos de Reportes.

Nombre del Servicio: Generar Gráficos de Reportes.

Versión: 1.0

Propietario*:* Proyecto SCADA

Tipo de servicio: Servicio de Negocio.

Requisito Funcional: RF3. Generar Gráficos de Reportes.

Capa**:** Servicio de Negocio.

Datos:

Mensaje de solicitud: Datos a reflejar en el gráfico.

Mensaje de respuesta: Imagen SVG.

Operaciones:

Generar Gráfico de Líneas.

Generar Gráfico de Barras.

Generar Gráfico de Pastel.

Invocación: http://url_del_servidor/cgi-bin/generateGraphic.cgi

Descripción: http://url_del_servidor/services/generateGraphic.wsdl

Documentación:

Servicio definido por la función encargada de generar gráficos de acuerdo con los parámetros que desea medir el usuario. Los gráficos pueden ser de tipo de Barras, Pastel, etc. Una vez se haya creado se serializa la imagen del mismo y se envía como respuesta al usuario.

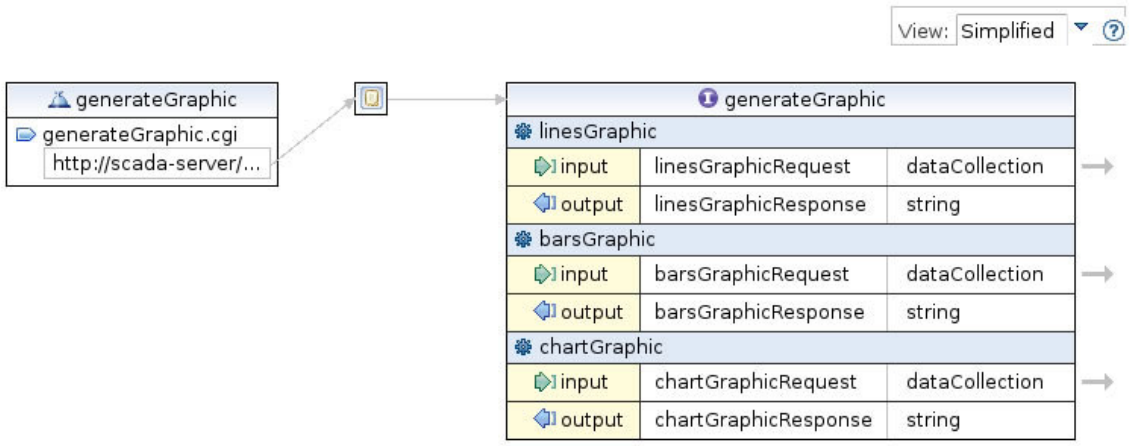


Figura: 3.3 Representación del contrato para el servicio Generar Gráfico.

De estas imágenes se deriva un el contrato para cada servicio. El ejemplo de código del mismo, en XML, se puede apreciar en el **Anexo 4** y **5** respectivamente.

Conclusiones.

En este capitulo se han abarcado las características fundamentales del sistema. Se describió la solución propuesta. Hubo una muestra de los artefactos resultantes en los Flujos de Trabajos: Modelo del Negocio y Captura de Requisitos. También se identificaron los servicios candidatos para el desarrollo de la solución.

En el siguiente capítulo se dará una visión detallada de la propuesta final del diseño al culminarse el ADOS.

Introducción

A continuación se abordan las actividades finales para el Análisis y Diseño Orientado a Servicios de los candidatos definidos en el capítulo anterior. Se muestran el modelo de servicios obtenidos en distintos diagramas, logrando con estos una perspectiva ideal para el posterior desarrollo de los servicios propuestos en este trabajo.

4.1 Modelo de Servicios

Llegado a este punto, lo primero para realizar este proceso es saber que necesita un servicio para que sea construido y que lógica encapsularía este como uno de los candidatos. En el capítulo anterior se creó la base necesaria para ello.

Para obtener un modelo de servicios utilizaremos RSA. A través de esta herramienta, UML propone una vía para diseñar servicios. En la siguiente plantilla (Fig. 4.1), se muestran las vistas recomendadas para el modelado, las cuales son aconsejables utilizar en la práctica profesional.

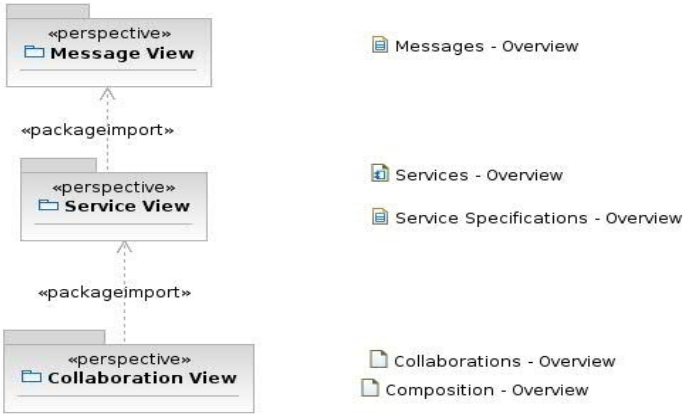


Figura: 4.1 Plantilla de Diseño de Servicios.

Para completar este proceso hay que tener en cuenta una serie de aspectos para una mejor confección de los artefactos a obtener.

- ✓ En primer lugar se identifican las particiones que se conforman con los servicios candidatos detectados. El objetivo de esto es agrupar los componentes, clases o nodos en las interacciones que ocurrirán cuando se requiera de las interfaces que expondremos.

- ✓ Luego se desarrollan las especificaciones de cada servicio. Estas contendrán la descripción de los métodos que se exponen, abstrayendo lo que hacen dichas funciones.

- ✓ En 3er lugar se diseñan los servicios. Aquí se muestran los proveedores de cada servicio y con ellos las interfaces, especificaciones y métodos que contienen. Esto da paso a la definición y diseño de los mensajes que tendrán lugar. Cada uno se conforma con datos de que son obtenidos de objetos existentes en la comunicación.

- ✓ Posterior a esto se hace la realización de los servicios. Aquí se obtienen diagramas detallados de las especificaciones para el desarrollo de los servicios.

4.2 Partición de los Servicios

Para realizar el proceso completo correctamente, primero se identifican las particiones de los servicios que encapsularán las funcionalidades que efectuarán las distintas operaciones que dan cumplimiento a una solicitud.

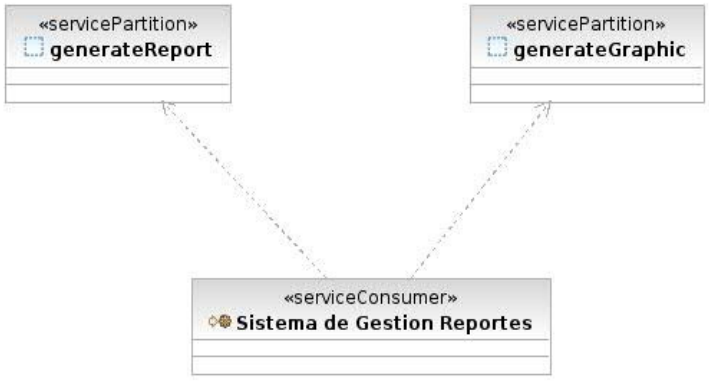


Figura: 4.2 Partición de los Servicios.

4.3 Patrón de diseño.

Cuando se diseñan servicios hay que tener en cuenta los principios de orientación a servicios. A partir de estos se determinan patrones de diseños a seguir. Analizando los principios más usados concluimos que:

- ☆ **Los límites son explícitos.**

El límite de un servicio no es más que el paso entre la interfaz pública y su implementación interna y privada. Esto queda claro una vez es publicado el WSDL, donde se plasma solo las funciones que se publican en la interfaz. Debido a esto el contrato del servicio se debe diseñar de modo que su evolución no implique la ruptura de contrato con los usuarios existentes.

El paso de límites puede ser costoso, porque la ubicación física del servicio puede ser un factor desconocido. Los modelos de seguridad y confianza pueden cambiar cada vez que se pasen estas limitaciones. Otro aspecto crítico es la activación y conversión de datos entre las representaciones públicas de un servicio que puede requerir el uso de recursos adicionales posiblemente externos al propio servicio. [**41**]

☆ **Los servicios deben ser autónomos.**

El diseño y desarrollo de los servicios deben permitir que no se conozcan las implementaciones de unos en otros. Los mismos serán implementados de forma independiente entre sí y su comunicación se realizará a través mensajes controlados por el contrato definido. Por tanto los contratos una vez que se publiquen no se podrán modificar. Por ello los desarrolladores deben elaborar sus esquemas de forma flexible. [**42**]

☆ **Los servicios comparten contratos, pero no clases.**

Los desarrolladores definen clases para representar entidades en el espacio de un problema. Estas clases combinan comportamiento y datos en una construcción en un solo lenguaje de programación con una plataforma explícita. Los servicios alteran este modelo con el fin de maximizar la flexibilidad y la interoperabilidad, llevando a cabo una comunicación mediante mensajes basados en XML son independientes de la plataforma y el lenguaje. [**43**]

4.3.1 Patrón Procesador de Documentos.

Los servicios de esta propuesta crean documentos en formatos determinados como respuesta a partir de una solicitud. Este patrón es aplicable a ello.

Se tiene en cuenta la creación de un contrato bien definido y con facilidad de uso. La respuesta a la solicitud del servicio es nomás que un archivo serializado. Dichas funcionalidades se encuentran encapsuladas en la implementación del servicio lo cual no deja escapar fuera de su límite las operaciones del mismo.

Con este patrón se mantiene la definición de los mensajes mediante un intercambio llevado a cabo por esquemas XML.

4.4 Paquetes de Diseño.

Un paquete de diseño es una colección de clases relacionadas, usadas para estructurar el modelo de diseño mediante su división en partes más pequeñas.

En el siguiente diagrama se muestran elementos del modelo de diseño, agrupados y relacionados con propósitos organizacionales.

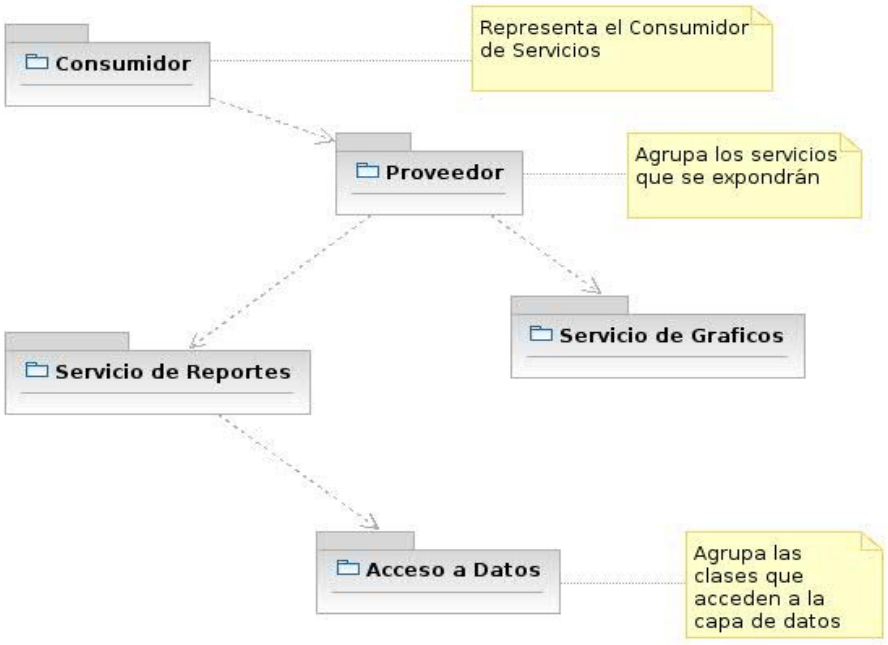


Figura: 4.3 Paquetes de Diseño.

4.5 Especificaciones de Servicios

Las especificaciones están dadas por los métodos que exponen los servicios. Estos encapsulan las operaciones que realmente realizan. Se utilizan para describir lo que hacen.

Figura: 4.4 Diagrama de Especificaciones de los Servicios.

4.6 Diseño de Servicios

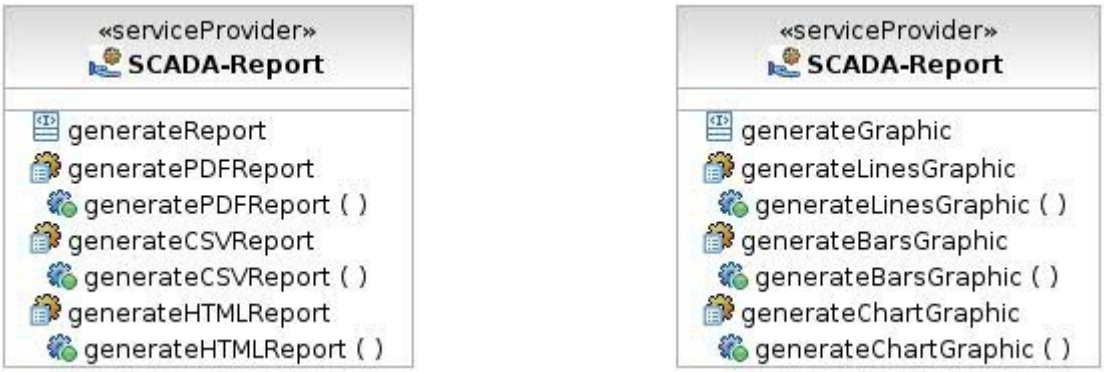


Figura: 4.5 Diagrama de Diseño de los Servicios.

En la figura se puede apreciar que el proveedor de ambos servicios es el mismo. Cada método mantiene su especificación y estos son agrupados en la interfaces que se exponen.

4.7 Diseño de Mensajes

En el diseño de los mensajes se tuvo en cuenta que parte de la información es compleja. Para cada interfaz se definió la misma estructura de mensajes, tanto de solicitud como de respuesta.

"generateReport Interface."

Figura: 4.6 Diagrama de Mensajes para generatePDFReport.

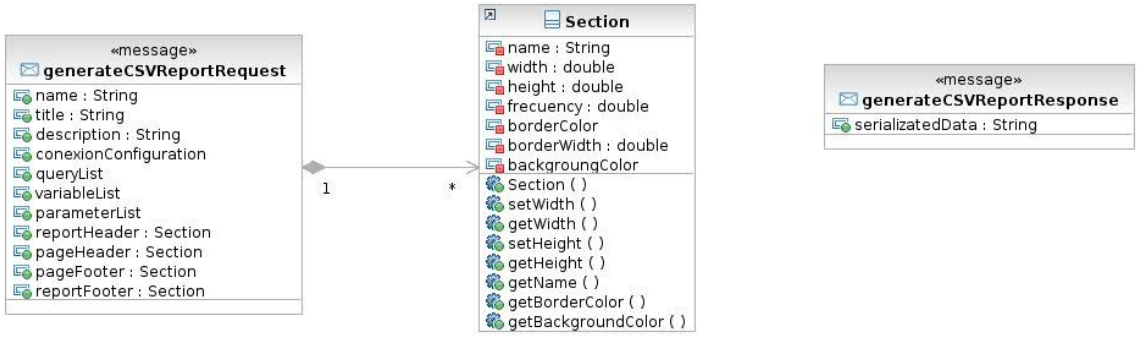


Figura: 4.7 Diagrama de Mensajes para generateCSVeport.

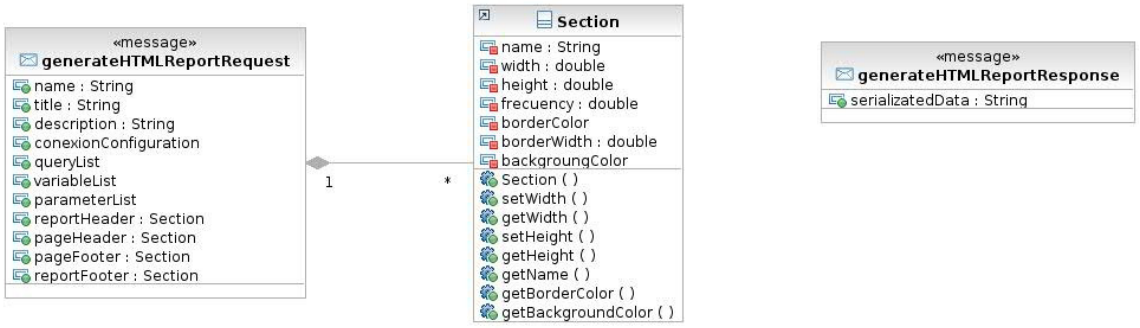


Figura: 4.8 Diagrama de Mensajes para generatePDFReport.

"generateGraphic Interface."

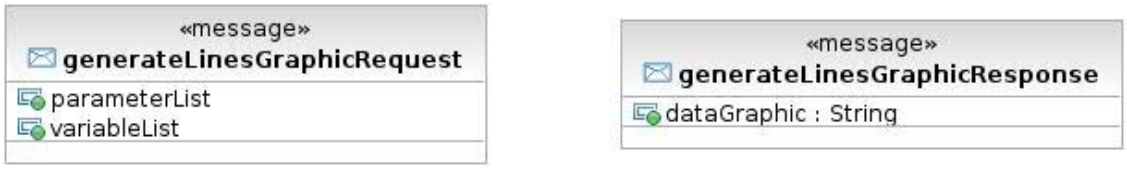


Figura: 4.9 Diagrama de Mensajes para generateLinesGraphic.

Figura: 4.10 Diagrama de Mensajes para generateBarsGraphic.

Figura: 4.11 Diagrama de Mensajes para generateChartGraphic.

Es importante destacar que los datos contenidos en los mensajes se tratan de la misma forma en cada interfaz. También se contempla una similitud en la mensajería lo cual indica que lo recibido por cada servicio es igual en cada especificación según su interfaz pero el resultado que se ofrece no es el mismo para ninguno de los casos.

Un ejemplo de esto se puede apreciar en los **Anexos 6** y **7,** donde para la misma entrada de datos se obtienen dos gráficas diferentes.

4.8 Realización de Servicios.

Estos diagramas abogan por la comprensión del funcionamiento de los servicios para su posterior desarrollo. En la realización de las especificaciones de esta interfaz se ha determinado un componente de acceso a dato para gestionar y controlar la información persistente.

Se puede apreciar que las funcionalidades están contenidas en un *namespace*, agrupando las clases y los elementos de la orientación a objetos.

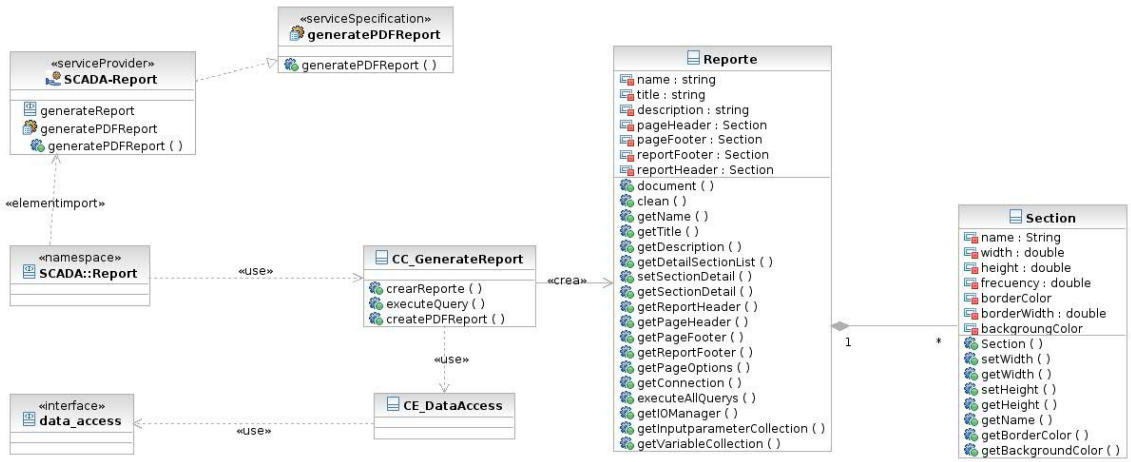


Figura: 4.12 Realización de la especificación generatePDFReport.

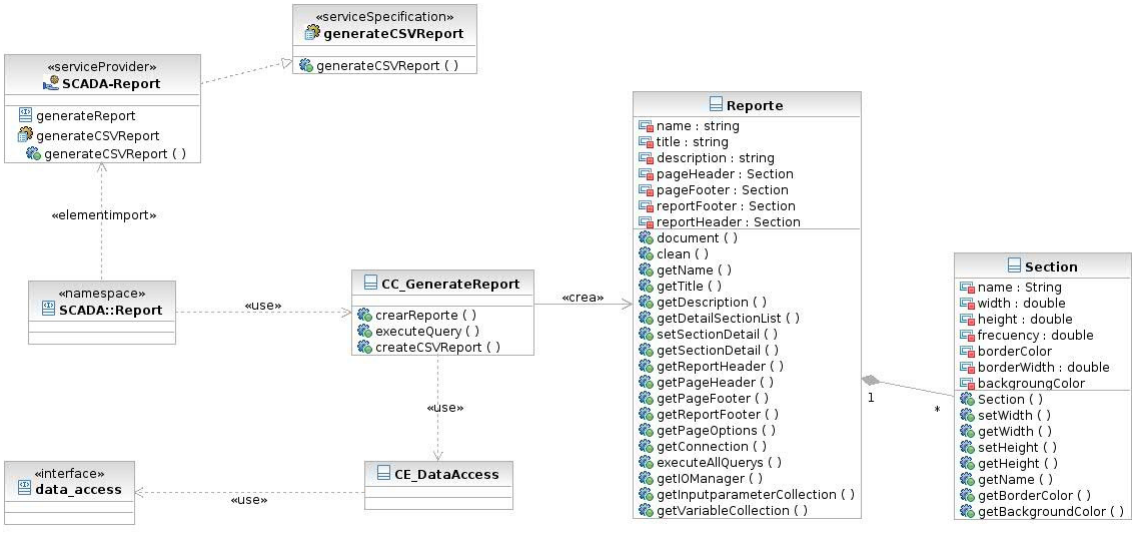


Figura: 4.13 Realización de la especificación generateCSVReport.

Figura: 4.14 Realización de la especificación generateHTMLReport.

Para los servicios de la interfaz generateGraphic no es necesario establecer componentes de acceso a datos para obtener la información necesaria de los datos persistentes. Estos servicios deben llamarse de forma asíncrona debido a la complejidad de las operaciones que contiene.

Los servicios de la interfaz generateGraphic tienen menor complejidad. Gracias al estándar SVG para imágenes es posible crear las imágenes deseadas en formato XML. Esto permite que las operaciones trabajen solo con datos básicos en cualquier lenguaje de programación.

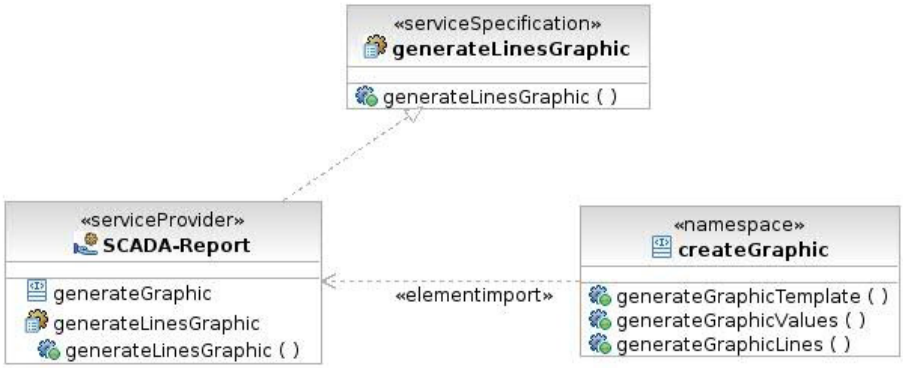


Figura: 4.15 Realización de la especificación generateLinesGraphic.

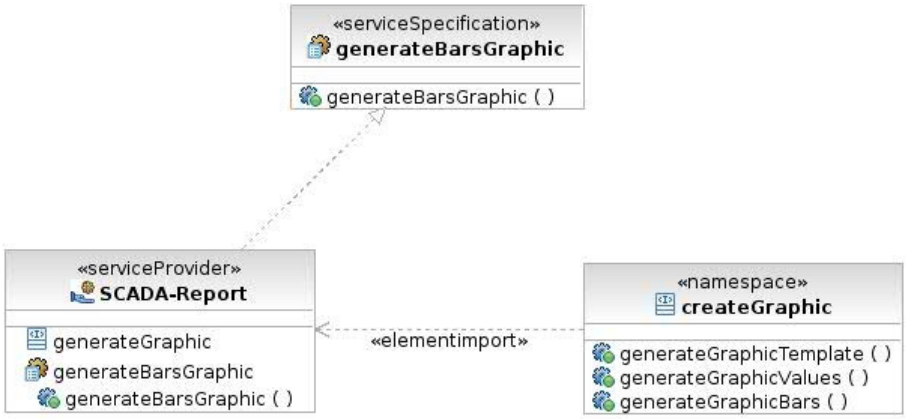


Figura: 4.16 Realización de la especificación generateBarGraphic.

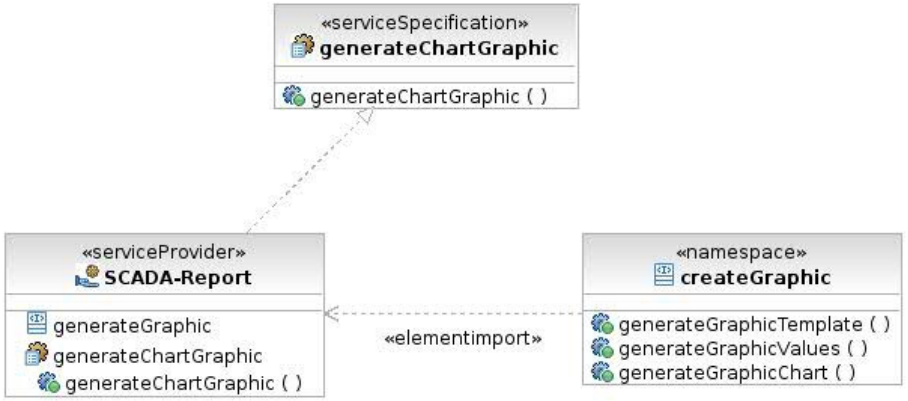


Figura: 4.17 Realización de la especificación generateChartGraphic.

4.9 Diagramas de Secuencia.

En estos diagramas de muestran una interacción ordenada según la secuencia temporal de los eventos existentes. Se muestran los objetos participantes y los mensajes que intercambian.

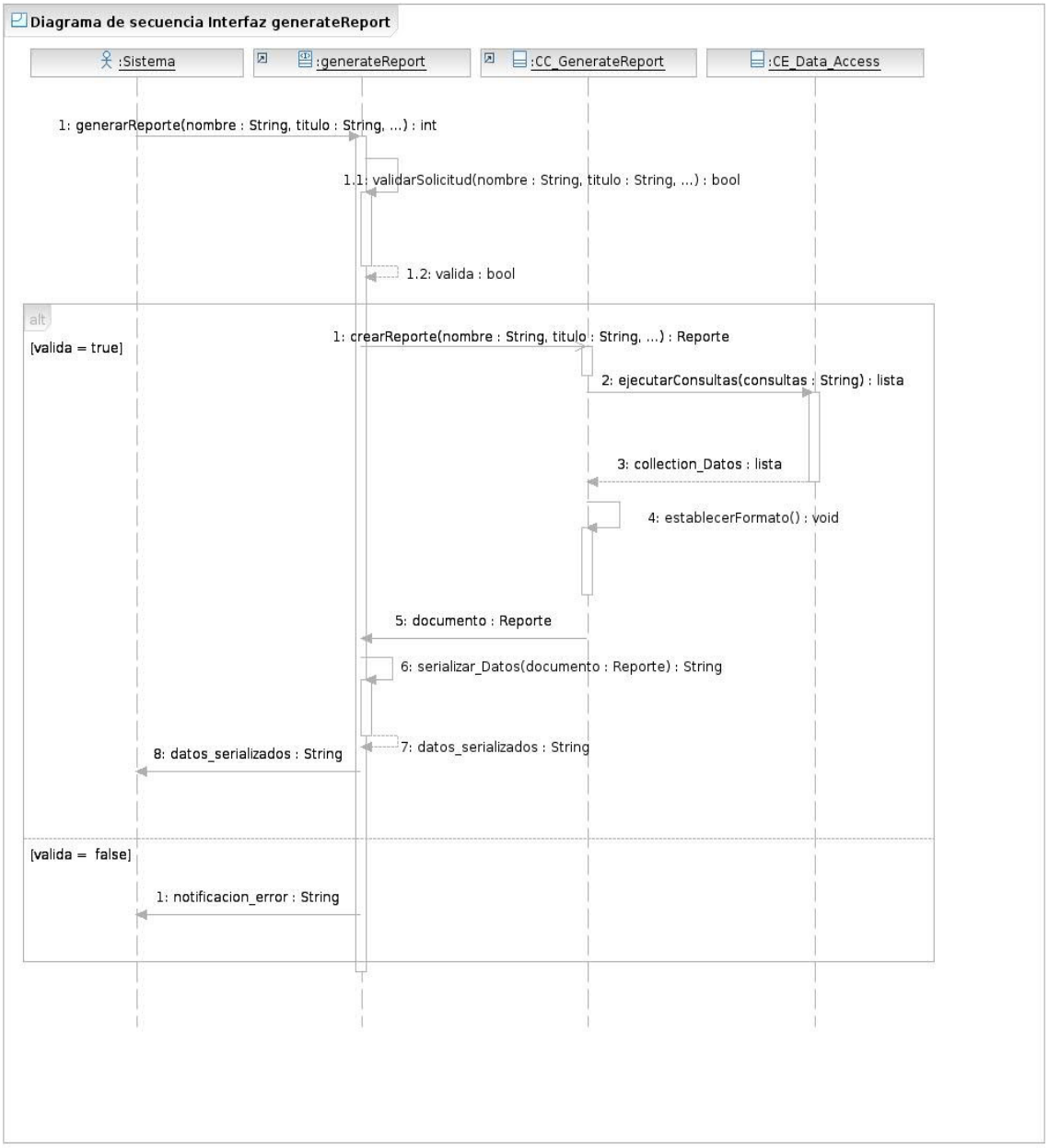


Figura: 4.15 Diagrama de secuencia Interfaz generateReport.

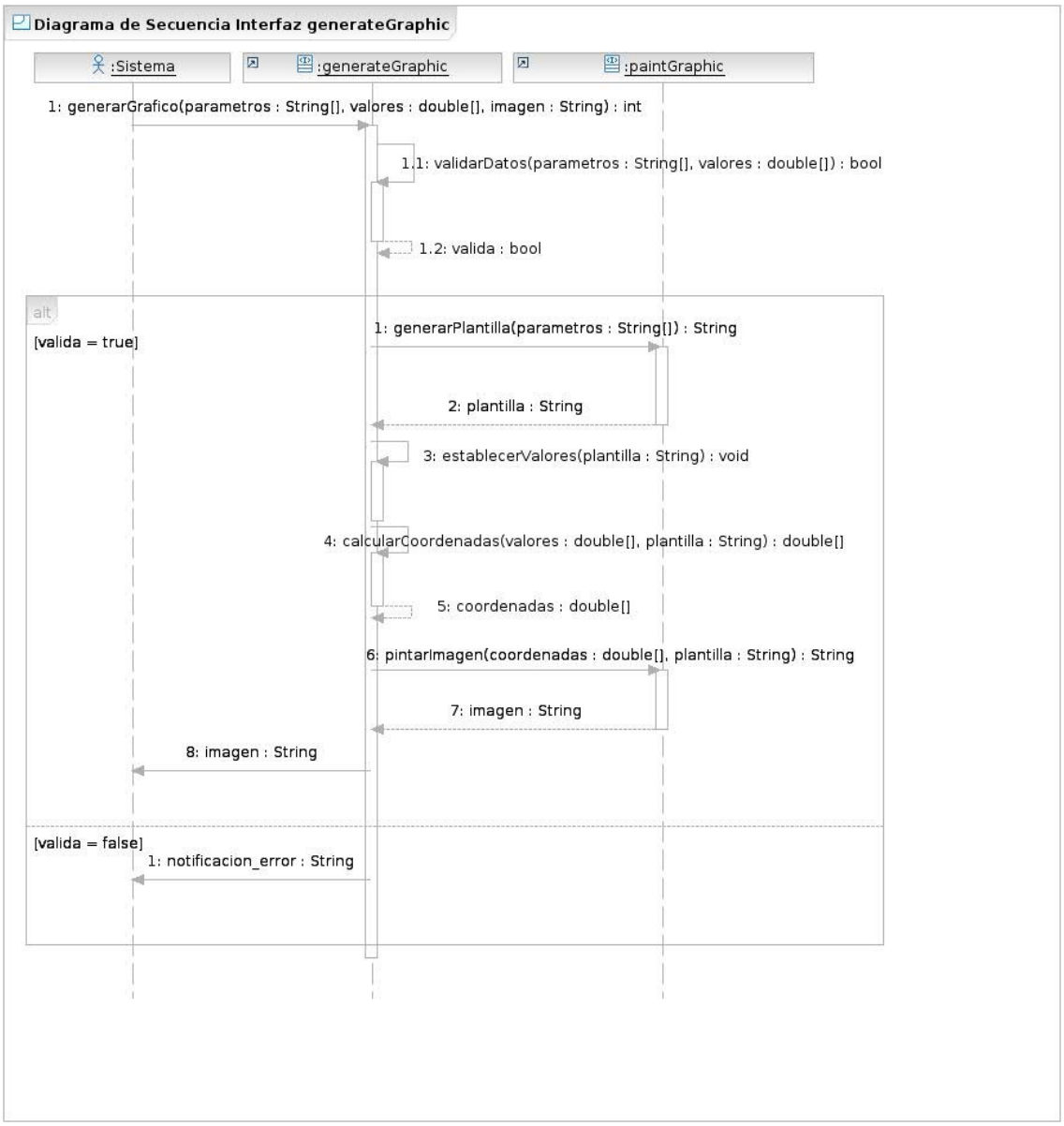


Figura: 4.16 Diagrama de secuencia Interfaz generateGraphic.

4.10 Diagrama de Despliegue.

Todo sistema SOA puede ser desarrollado usando una tecnología cliente – servidor. Esta propuesta siguiendo la misma metodología y basándose en lo antes expuesto presenta el siguiente diagrama de despliegue. Se distribuye la tecnología en distintos nodos que contienen los elementos de procesamiento, y

las conexiones entre estos. De la siguiente forma quedaría la distribución de los componentes físicos.

En este caso la comunicación entre el consumidor y el proveedor de servicios será mediante el protocolo SOAP, usando el lenguaje WSDL para describir las funciones que expone la interfaz de servicios. Esta última usa las especificaciones contenidas en las particiones (Abstracción de las interfaces y especificaciones), donde una de ellas necesitará conexión a la BD mediante la tecnología ADO.

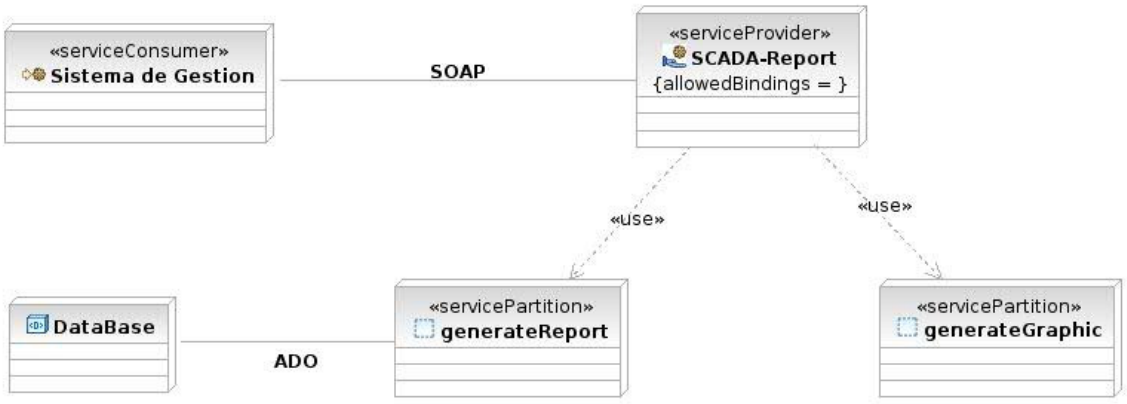


Figura: 4.17 Diagrama de Despliegue para la solución propuesta.

4.11 Descripción del prototipo

Para complementar todo lo hasta aquí analizado se desarrolló un prototipo funcional para tener una visión mas específica del producto que se propone obtener como solución. El prototipo esta basado en la funcionalidad del servicio generateGraphic.

Para modelar el contrato del mismo se realizan las operaciones que se definen a continuación:

- Se definen los mensajes que se contendrán dentro del límite del servicio.

- Se definen las operaciones que realiza el servicio para dar respuesta a las solicitudes de los consumidores.

- Se agrupan las operaciones que se exponen en la interfaz del servicio como procesos públicos, las cuales reflejan los mensajes (datos públicos) necesarios para establecer la comunicación de intercambio de datos.

graphic.h

```
1//gsoap ns service name:            generateGraphic
2//gsoap ns service style:           rpc
3//gsoap ns service location:        http://10.35.1.13/cgi-bin/service/generateGraphic.cgi
4//gsoap ns service namespace:       http://10.35.1.13/generateGraphic.wsdl
5
6//gsoap ns schema namespace:        urn:generateGraphic
7
8//gsoap ns service method-documentation: Funcion que genera gráficos estadisticos con formato de linea.
9int ns__lineGraphic(char ** params, double *values, char *&graphic);
10
11//gsoap ns service method-documentation: Funcion que genera gráficos estadisticos con formato de barras.
12int ns__barGraphic(char ** params, double *values, char *&graphic);
13
```

Figura: 4.18 Definición del Contrato para al Interfaz generateGraphic.

Una vez definido el contrato se crea el skeleton o stubs para el cliente y el servidor. Se ejecuta la herramienta soapcpp2 seguida del archivo que contiene el contrato. Los archivos obtenidos se separan y se completa la implementación para el consumidor y para el proveedor.

El consumidor o cliente conforma los datos contemplados en el contrato para la comunicación con el servidor. El proveedor se instala como CGI en un servidor Apache. La dirección física del servicio debe quedar contemplada en el contrato.

Tras invocar desde el consumidor la solicitud con una serie de datos se generan dos tipos de respuestas diferentes, las imágenes generadas pueden ser apreciadas en los Anexos 6 y 7.

Conclusiones.

En este capítulo hemos plasmado todos los elementos necesarios para dar entrada a la construcción de los servicios que satisfacen los requisitos funcionales del sistema. En los diagramas expuestos se tuvo en cuenta el refinamiento de los requisitos no funcionales. Hasta aquí se ha podido dar una visión más clara para el desarrollo de nuestro sistema.

Diseñar servicios puede resultar un proceso complejo. En este capítulo se ha abordado más a fondo la ingeniería de la orientación a servicio. Gracias a los criterios estándares y a la propuesta que brinda la herramienta IBM RSA mediante UML 2.0 para esto, la propuesta a la solución del problema es factible y satisface lo requerido.

CONCLUSIONES

Con el nuevo diseño realizado para el Módulo de Reportes se da cumplimiento al objetivo de este trabajo, pues se obtuvo un modelamiento del software en el que se aplican los resultados de la investigación realizada. Los logros más relevantes se mencionan a continuación:

- ☆ Se obtuvo una nuevo diseño del software proporcionando una entrada óptima al desarrollo de este sistema SOA el cual cuenta con 2 servicios encargados de generar los reportes.
- ☆ Se propone una tecnología para el desarrollo de la aplicación y se demostró con un prototipo funcional que es posible la construcción de la misma.
- ☆ Se efectuó una comparación entre la arquitectura actual y la que se aplicará en la solución con el fin de aprovechar elementos de distintos paradigmas para que el trabajo sea óptimo.
- ☆ Gracias a la estrategia adoptada la gestión de cambios es más flexible y eficiente. Con ella se mantiene en constante interacción la evolución del negocio y el desarrollo de la solución y posible incremento de esta última.
- ☆ El módulo de reportes del SCADA cuenta con arquitectura que atribuye las ventajas de la misma.

RECOMENDACIONES

Tomando como base la investigación realizada y los resultados obtenidos durante la realización de este trabajo, se recomienda lo siguiente:

- ☆ Aplicar este trabajo como solución al problema existente en el módulo de reporte del SCADA.
- ☆ Mejorar el prototipo funcional e integrarlo al módulo de reportes.
- ☆ Desarrollar el servicio de generar reportes como parte de la solución.
- ☆ Desarrollar un servicio de seguridad para controlar el acceso a estos servicios y así proteger la información que aquí se manipula.

REFERENCIAS BIBLIOGRÁFICAS

1. www.everis.es.
2. wadooa.com/doku.php/soa.
3. arquitecturaorientadaaservicios.blogspot.com/2006/03/pero-qu-es-realmente-soa.html.
4. Microsoft Corporation, 2006, Service Oriented Architecture (SOA) in the Real World, Cap 1.
5. www.espaciosoa.net/2008/03/03/soa-saas-y-otros-menesteres/.
6. en.wikipedia.org/wiki/Composite_application.
7. en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29.
8. bussinesssoft.blogspot.com/2007/09/qu-es-soa.html.
9. lucasian.com/soa/2007/07/31/sistemas-empresariales-bajamente-acoplados-con-arquitectura-soa/
10. www-306.ibm.com/e-business/la/ar/soa_site/soa_2.shtml.
11. James McGovern, Morgan Kaufman y otros, 2003. Java Web Services Architecture, Cap. 2.
12. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cap 8
13. Ídem 12.
14. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cap 4
15. www.informationweek.com.mx/articulo-46-6855-398.html

16. Norbert Bieberstein, Sanjay Bose, Marc Fiammante, Keith Jones, Rawn Shah october 19, 2005, Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap, Cp 9.

17. Ídem 16.

18. Ídem 16.

19. www.emb.cl/gerencia/articulo.mv?sec=1&num=185

20. geeks.ms/blogs/ciin/archive/2007/10/15/service-oriented-architecture-soa-191-por-d-243-nde-empezar.aspx

21. Ídem 19 y 20.

22. www.sun.com/emrkt/innercircle/newsletter/latam/1006latam_sponsor.html

23. Ídem 1 y 22.

24. www.infotechnology.com/notas/121096-se-va-la-segunda

25. Ídem 22 y 24.

26. Alberto Cubo Velázquez. “Representational State Transfer (REST). Un estilo de arquitectura para Servicios Web. Panorámica y estado del arte.” 2006.

27. Olaf Zimmermann, Pal Krogdahl, Clive Gee, Elements of Service-Oriented Analysis and Design, june 2, 2004, www.ibm.com/developerworks/webservices/library/ws-soad1/#N102DB

28. Raju Cherukuri, www.ibm.com/developerworks/architecture/, april 21, 2008.

29. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cap 11.

30. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cap 13

31. IBM – Developer Works, Elements of Service-oriented Analysis and Design", june 2004.

32. Endrei M., "Patterns: Service-oriented Architecture and Web Services", april 2004 Redbook.

33. Ali Arsanjani, Service - Oriented modeling and architecture, noviembre 2, 2004, http://www.ibm.com/developerworks/architecture/

34. Martin Keen, Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, Paul Verschueren. "Patterns: Implementing an SOA with the Enterprise Service Bus", august 2004, Redbook.

35. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cap 10, sección 10.1.

36. Ídem 35.

37. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cp 10, section 10.2.

38. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cp 10, section 10.3.

39. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cp 10, section 10.4.

40. Thomas Erl, august 04, 2005, Service-Oriented Architecture: Concepts, Technology, and Design, Cp 11.

41. John Evdemon, Principios de Diseño de servicios: Patrones y antipatrones de servicios. diciembre 12, 2005.

http://www.microsoft.com/spanish/msdn/articulos/archivo/121205/voices/SOADesign.mspx

42. Ídem 41.

43. Ídem 41.

44. IBM Corporation Software Group, diciembre 2006, ibm.com/rational/adc.

45. Learn About Service Oriented Architecture, http://www.microsoft.com/biztalk/solutions/soa/default.mspx, diciembre 1, 2006.

BIBLIOGRAFÍA

1. Wikipedia, Arquitectura Orientada a Servicios (SOA), http://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios.

2. Navactiva, octubre 17, 2005. SOA e Integracion de la Informacion Corporativa. http://www.navactiva.com/web/es/avtec/doc/informes/2005/10/34134.php.

3. BlogSpot, mayo 9, 2006, Arquitectura Orientada a Servicios (SOA), http://arquitecturaorientadaaservicios.blogspot.com/2006/05/elementos-esenciales-de-una.html.

4. BlogSpot, junio 6, 2006, Arquitectura Orientada a Servicios (SOA), http://arquitecturaorientadaaservicios.blogspot.com/2006/06/soa-y-los-servicios-web-i.html.

5. BlogSpot, junio 9, 2006, Arquitectura Orientada a Servicios (SOA), http://arquitecturaorientadaaservicios.blogspot.com/2006/06/soa-y-los-servicios-web-ii.html.

6. BlogSpot, junio 22, 2006, Arquitectura Orientada a Servicios (SOA), http://arquitecturaorientadaaservicios.blogspot.com/2006/06/principios-de-la-orientacin-servicios.html.

7. Javier Cámara, mayo 23, 2007, SOA y estándares: una pareja inseparable.

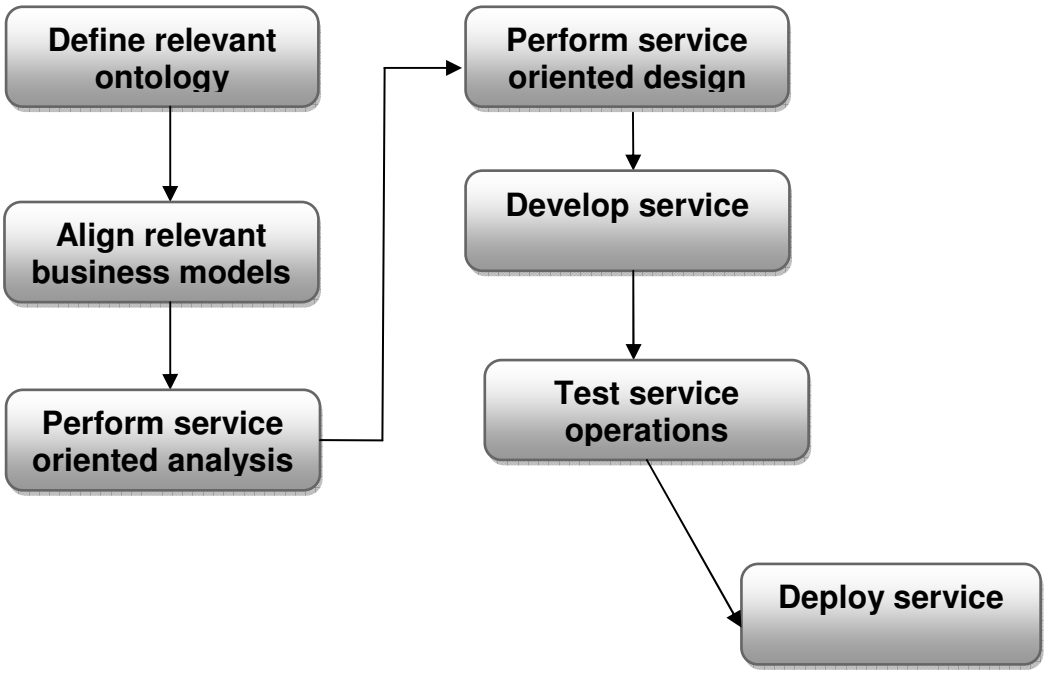
8. Roy W. Schulte, Gartner, diciembre 9, 2002.Predicts 2003: SOA is changing software.

9. Yefim V. Natis, Gartner, april 16, 2003. Service-Oriented Architecture Scenario.

10. Huibert Aalbers, Elección de tecnología para la capa de presentación de SOA, mayo 4, 2008, http://www.huibert-aalbers.com.

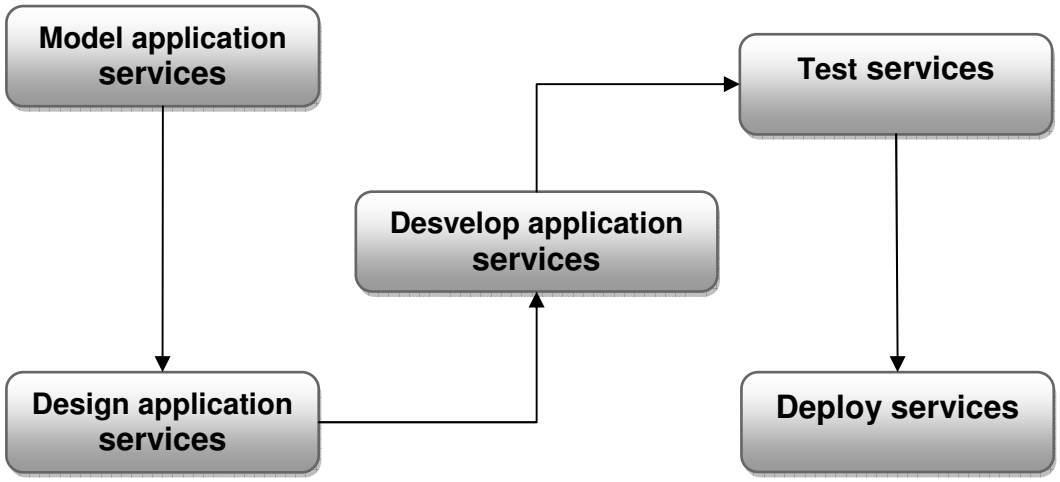
ANEXO 1:

FiguraA1: Pasos de la Estrategia Top – Down.



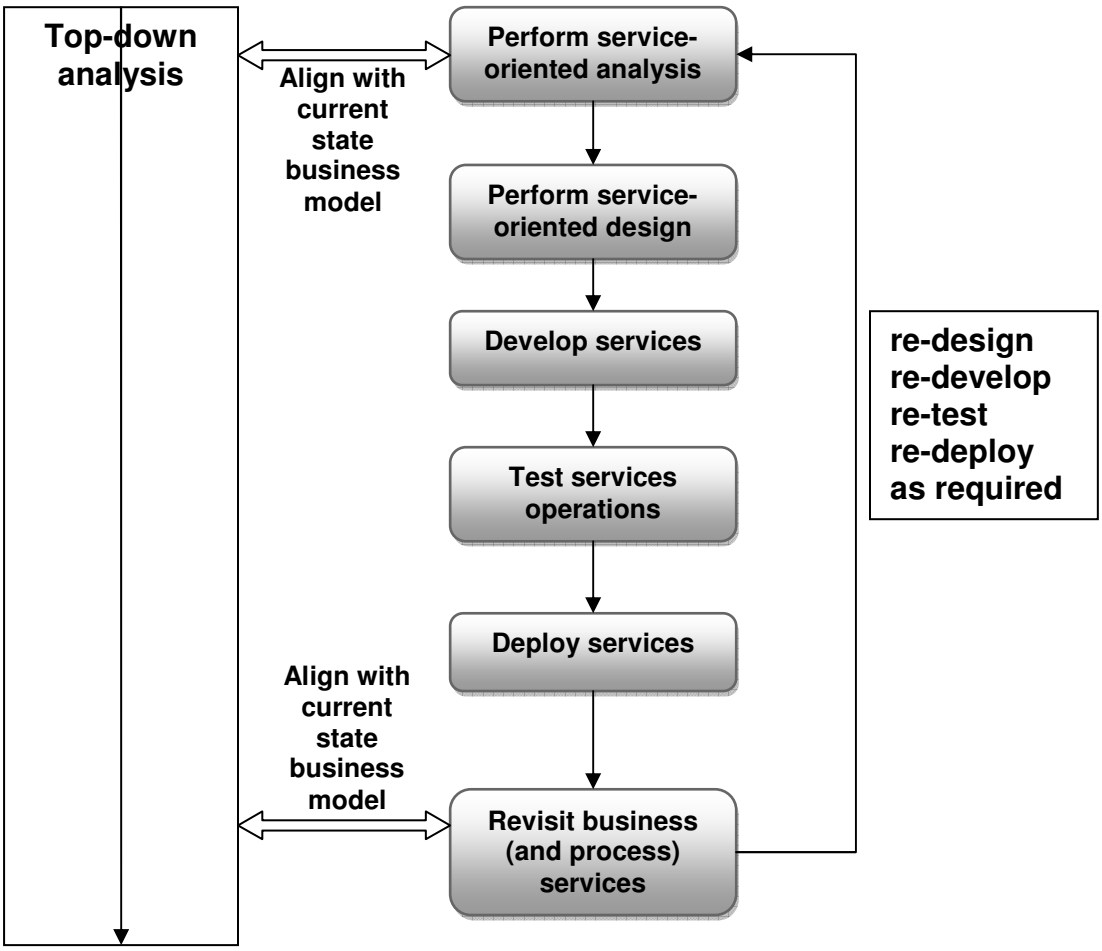
ANEXO 2:

Figura A2: Pasos de la Estrategia Bottom – Up.



ANEXO 3:

Figura A3: Pasos de la Estrategia Agile (meet-in-the-middle).



Anexo 4:

Fragmento ejemplo del WSDL para el servicio generateReport.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="generateReport"
targetNamespace="http://www.example.org/NewWSDLFile/"
xmlns:tns="http://www.example.org/NewWSDLFile/">
    <wsdl:types><xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/NewWSDLFile/">
(...)
  <wsdl:portType name="generateReport">
    <wsdl:operation name="generateReport">
```

```
      <wsdl:input message="tns:generateReportRequest"/>
      <wsdl:output message="tns:generateReportResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="generateReport" type="tns:generateReport">
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="generateReport">
      <soap:operation
soapAction="http://www.example.org/NewWSDLFile/NewOperation"/>
      <wsdl:input>
        <soap:body namespace="http://www.example.org/NewWSDLFile/"
use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body namespace="http://www.example.org/NewWSDLFile/"
use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="generateReport">
    <wsdl:port binding="tns:generateReport" name="generateReport.cgi">
      <soap:address location="http://scada-server/cgi-bin/service/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Anexo 5:

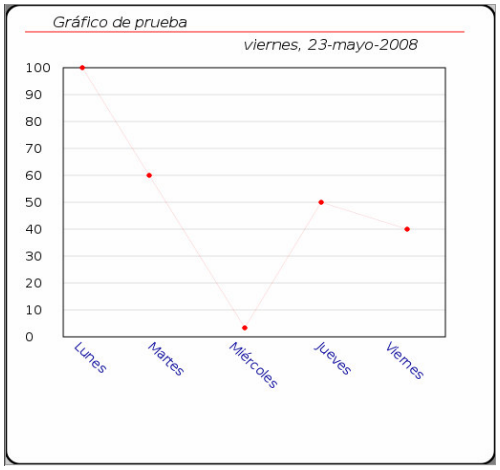
Fragmento ejemplo del WSDL para el servicio generateReport.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.example.org/generateGraphic/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="generateGraphic"
targetNamespace="http://www.example.org/generateGraphic/">
    <wsdl:types><xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/generateGraphic/">
(...)
  <wsdl:portType name="generateGraphic">
    <wsdl:operation name="linesGraphic">
      <wsdl:input message="tns:linesGraphicRequest"/>
      <wsdl:output message="tns:linesGraphicResponse"/>
    </wsdl:operation>
    <wsdl:operation name="barsGraphic">
        <wsdl:input message="tns:barsGraphicRequest"></wsdl:input>
        <wsdl:output message="tns:barsGraphicResponse"></wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="chartGraphic">
        <wsdl:input message="tns:chartGraphicRequest"></wsdl:input>
        <wsdl:output message="tns:chartGraphicResponse"></wsdl:output>
```

```
    </wsdl:operation>
  </wsdl:portType>
(...)
  <wsdl:service name="generateGraphic">
    <wsdl:port binding="tns:generateGraphicSOAP"
name="generateGraphic.cgi">
      <soap:address location="http://scada-server/cgi-bin/service/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

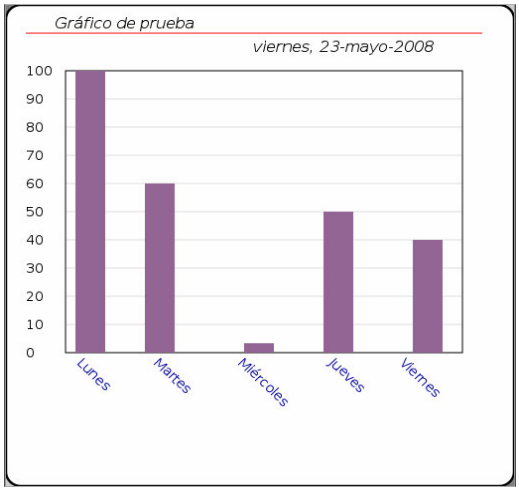
Anexo 6:

Figura A4: Gráfico generado por el servicio generateGraphic.



Anexo 7:

Figura A5: Gráfico generado por el servicio generateGraphic.



GLOSARIO

ADOS: Análisis y Diseño Orientado a Servicios, SOAD (Service – Oriented Analysis and Design) en inglés. Metodología para modelar una aplicación basada en orientación a servicios.

Apache: Apache es programa de servidor HTTP Web de código abierto (. Fue desarrollado en 1995 y actualmente es uno de los servidores web más utilizados en la red. Usualmente corre en UNIX, Linux, BSD y Windows. Es un poderoso paquete de servidor web con muchos módulos que se le pueden agregar y que se consiguen gratuitamente en el Internet.

***Application Programming Interface (API)*:** Conjunto de rutinas y definiciones de funciones que abstraen los detalles de la implementación y hace más fácil el desarrollo de aplicaciones.

ASP: Acrónimo en inglés de Active Server Pages. Páginas de Servidor Activo. Son un tipo de HTML que además de contener los códigos y etiquetas tradicionales, cuenta con programas (o scripts) que se ejecutan en un servidor Microsoft Internet Information Server antes de que se desplieguen en la pantalla del usuario.

***Business Process Management (BPM)*:** Administrador de Procesos de Negocio: metodología empresarial se gestión sistemática de los procesos de negocio.

C / C++: C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel.

Common Gateway Interface (CGI): Mecanismo de comunicación cliente servidor. Las aplicaciones que se ejecutan en un servidor Web reciben el nombre de CGI.

Component Object Model (COM): Interfaz estándar para la composición de software, introducida por Microsoft en 1993. Usada para establecer comunicación entre procesos mediante objetos dinámicos de cualquier lenguaje de programación que soportara esta tecnología.

***Contrato*:** Operaciones que expone un servicio, datos que recibe y que retorna cada operación.

Common Object Request Broker Architecture (***CORBA):*** Conjunto de estándares industriales publicados por la OMG, que define un modelo distribuidos para sistemas de objeto de aplicación.

Enterprise Service Bus (ESB): Bus de Servicios de Empresa: capaz de conectar y coordinar cientos de aplicaciones mediante la combinación de Servicios Web y XML.

***Firewall*:** Cortafuego. Como el firewall de Windows. Herramienta del sistema operativo que mantiene la seguridad de acceso a través de la red a las computadoras.

Free Software Fundation (FSF): Entidad que busca eliminar las restricciones de uso, copia, modificación y distribución del software. Apoya el desarrollo de sistemas operativos (GNU/Linux), compilador GNU C (GCC), PERL, etc.

GNU/Linux: Sistema operativo libre creado por Richard M. Stallman y Linus Torvals que en la actualidad cuenta con varias distribuciones o versiones.

General Public License (GPL): Esta licencia regula los derechos de autor de los programas de software libre (free software) promovido por la FSF en el marco de la iniciativa GNU.

Hypertext Markup Language (HTML): Es un lenguaje para crear documentos de hipertexto para uso en el WWW o intranets, por ejemplo. Los archivos de HTML son usualmente visualizados por navegadores como Internet Explorer,

Firefox y Safari, entre otros. Es independiente del sistema operativo de la computadora.

Hypertext Transfer Protocol (HTTP): Es un protocolo con la ligereza y velocidad necesaria para distribuir y manejar sistemas de información hipermedia. HTTP ha sido usado por los servidores World Wide Web desde su inicio en 1993.

IBM Rational Software Architect (RSA): Herramieta integrada para diseñar y desarrollar modelos con UML.

Integrated Development Environment (IDE): Software que provee facilidades a los desarrolladores en lenguaje de programación y compatibilidad con el sistema operativo o plataforma en que se trabaje.

Interfaz de servicios: Entidad de software implementada normalmente como una fachada que expone métodos, a los que se puede llamar de forma individual o en una secuencia específica para formar una conversación que implemente una tarea.

Mensajes*:* Mecanismo de comunicación entre los servicios. Los datos que componen el mismo están contenidos en formato XML Schema.

Multipurpose Internet Mail Extension (MIME): Sistema que permite integrar dentro de un mensaje de correo electrónico ficheros binarios (imágenes, sonido, programas ejecutables, etc.).

Object Management Group (OMG): Grupo que produce y brinda mantenimiento a computadoras con fines industriales para aplicaciones empresariales interoperables.

Object Oriented Programming (OOP): Programación Orientada a Objetos.

Plug-in: Funcionalidad que se le puede agregar a un programa

Portable Document Format (PDF): Formato de Documento Portable, formato gráfico creado por la empresa Adobe el cual reproduce cualquier tipo de documento en forma digital idéntica, facsímil, permitiendo así la distribución electrónica de los mismos a través de la red en forma de archivos PDF.

***Repositorio de servicios*:** Servicio que contiene la dirección en que están los proveedores de servicios de un sistema SOA.

Remote Method Invocation (RMI): Protocolo RPC publicado por Sun Microsystems para acceder remotamente a funciones Java dentro de un sistema distribuido.

***Remote Procedure Call (RPC)*:** (Remote Procedure Call, por sus siglas en inglés). Llamada a Procedimiento Remoto: un programa en una computadora puede correr un programa en otra computadora**.**

Rational Unified Process (RUP): Proceso unificado de desarrollo. Proceso de ingeniería de un software desarrollado por IBM Rational. Incorpora las mejores prácticas de la ingeniería.

Service – Level Agreement (SLA): Es el contrato entre un proveedor y un consumidor de servicio que describe el comportamiento del mismo.

***Servicios*:** Componente o programa con el que interactuamos intercambiando mensajes.

***Service – Oriented Architecture (SOA)*:** Abreviatura de Arquitectura Orientada a Servicios.

Simple Object Access Protocol (SOAP): Protocolo basado en mensajería XML definido por W3C. Usado para codificar la información tanto de solicitud como de respuesta en los servicios Web.

Sun Microsystems: Empresa ubicada en Mountain View, California, que fabrica hardware y software para computadoras. Conocida por crear programas para UNIX. En los últimos años ha desarrollado el lenguaje de programación Java.

Tecnología de Información (TI): Término muy general que se refiere al campo entero de la tecnología informática - que incluye hardware de computadoras y programación hasta administración de redes.

Universal Description, Discovery and Integration (UDDI): Protocolo que permite descubrir servicios web existentes en Internet y conocer su descripción para saber qué tarea realizan.

Web Service: Familia de tecnologías conformada por descripciones, protocolo y estándares de la industria usados para la comunicación entre aplicaciones heterogéneas.

Web Services Description Languaje (WSDL): Describe la manera adecuada de utilizar un servicio web desde otros programas.

World Wide Web (WWW): Es el sistema de información basado en hipertexto, cuya función es buscar y tener acceso a documentos a través de la red de forma que un usuario pueda acceder usando un navegador web.

World Wide Web Consortium (W3C): Consorcio internacional, creado por Tim Berners-Lee en 1994, en donde se desarrollan estándares y especificaciones relacionados al WWW.

eXtended Markup Lenguaje (XML): Lenguaje de Marcas que permite que el usuario defina sus propias etiquetas para los documentos de intercambio de información, a diferencia de HTML en que son fijas.

XML Schema: Lenguaje para definición de los tipos de datos contenidos en un documento XML. Permite luego validar que un document XML está bien construido. Permite utilizar las mismas tecnologías para su uso.