Pontificia Universidad Católica de Valparaíso

Facultad de Ingeniería

Escuela de Ingeniería Informática

ANALISIS Y DISEÑO DE UNA ARQUITECTURA SOA PARA UNA INSTITUCIÓN FINANCIERA

Autor:

Cristian Andres Mohor Tapia

Memoria para optar al Título profesional de

Ingeniero Civil en Informática

Profesor Guía:

Jorge Bozo Parraguez

DICIEMBRE – 2006

“Dedico esta Memoria a Dios.
A mi Familia, por su apoyo incondicional.
A la Universidad, Escuela y Profesores.”

Cristian Andrés Mohor Tapia

Agradecimiento

“Agradezco a todos los que estuvieron apoyándome en esta etapa, en especial a mi madre y familia, por entregarme la fortaleza para seguir este viaje.

A mi esposa, por darme la fuerza para terminar, y así empezar una nueva etapa juntos.

Por último, quiero agradecer a los profesores, por ser parte de mi formación profesional”

Cristian Andrés Mohor Tapia.

Resumen.

SOA es un modelo arquitectónico que permite establecer un orden y un marco de trabajo para los sistemas de información, facilitando la reutilización de código e integración entre procesos; disminuyendo la tasa de errores producida por la redundancia de datos y entregado resultados en formatos estándares. Junto con SOA se encuentran los Web Services (WS), que corresponden a servicios cuya principal característica es su forma de

comunicación por un canal Web, utilizando un lenguaje estándar (WSDL) y un protocolo de comunicación definido (SOAP).

Dentro de este ámbito, la presente memoria de título comienza con una investigación conceptual sobre la Arquitectura Orientada a Servicios (SOA) y los Web Services (WS), cuyo resultado se traduce en la elaboración de una arquitectura SOA para una institución financiera. Este caso práctico, permite ver el funcionamiento de dos servicios. El primero de ellos integra los datos de un cliente asociado a una póliza de seguros entre dos sistemas operacionales; mientras que el segundo servicio, actualiza los datos para un mandato (documento utilizado por un cliente para gestionar un pago automático hacia uno o varios de sus productos).

Palabras claves: Arquitectura Orientada a Servicios (SOA), integración, institución financiera, sistemas de información.

Abstract.

SOA is an architectonic model that allows to establish an order and a frame of work for the information systems, being facilitated the reusability of code and integration between processes; diminishing the error rate produced by the redundancy of data and given results in standard formats. Along with SOA are the Web Services (WS), that correspond to services whose main characteristic is its form of communication by a channel Web, using a standard language (WSDL) and a defined communication protocol (SOAP).

Within this scope, the present memory of title begins with a conceptual investigation on the Oriented Architecture to Servicios (SOA) and the Web Services (WS), whose result is translated in the elaboration of an architecture SOA for a financial institution. This practical case, allows to see the operation of two services. The first of them includes the data of a client associated to an insurance policy between two operational systems; whereas the second service, updates the data for a mandate (document used by a client to demand an automatic payment towards one or several of its products).

Keywords: Architecture oriented to services (SOA), integration, financial institution, information systems.

Introducción

En la actualidad las empresas cuentan con sistemas de información a medida, desarrollados para apoyar a cada una de las áreas de negocios especifica, cuyo origen, en muchos casos, corresponde a soluciones implementadas a través de diversas tecnologías, generando verdaderas islas en relación a la administración de recursos e información. El inconveniente surge cuando se requiere operar entre sistemas de distintas áreas en pos de un objetivo común. En estos casos, es frecuente que las empresas dispongan de procesos de integración que permitan generar una comunicación directa entre los grandes sistemas que manejan el *core* del negocio, con el fin de obtener, por ejemplo, vistas de la situación real de los procesos operativos que soportan los sistemas de información, componentes claves en las tareas de administración y toma de decisiones por parte de los directivos.

En este punto cobra importancia el hecho de proveer una arquitectura de software que sea transversal a la organización, enfocada a desarrollar una estrategia de aplicaciones empresariales que facilite su integración, motivando la construcción de servicios, más que de aplicaciones. De esta manera, una aplicación final simplemente administra la ejecución de un conjunto de estos servicios, añade su lógica particular y le presenta una interfaz al usuario final en forma estándar.

En respuesta a estas necesidades nace el concepto de SOA (*Service Oriented Architecture*), que entre otras arquitecturas como CORBA (C*ommon Object Request Broker Architecture*) y DCOM (*Distributed Component Object Model*) logran proveer una funcionalidad similar, pero con la ventaja de ser una arquitectura multiplataforma y con una alta capacidad de reutilización.

La arquitectura SOA propone un modelo mucho más eficiente, donde el código de la función es independiente de la forma en que se resuelve la integración. La función puede estar hecha en cualquier lenguaje de programación y residir en cualquier tipo de plataforma tecnológica. Una arquitectura SOA está formada por tres partes: un proveedor, un intermediario y un cliente que no presentan ningún acoplamiento entre ellos. El proveedor

ofrece un servicio determinado y que el cliente no tiene porque conocer directamente. El cliente aprende como utilizar el servicio a partir de la información que le ofrece el intermediario que normalmente simplifica el uso de dicho servicio. El cliente sólo sabe como utilizar el servicio, es decir, como enviar y recibir datos pero no conoce ningún detalle de su implementación interna. La integración se resuelve mediante una definición clara de los parámetros de llamada a los servicios y una definición específica de la naturaleza de la respuesta. Un ejemplo típico de arquitectura SOA son los *Web Services* (Servicios Web) que proporcionan una interfaz de acceso a un servicio escondiendo las particularidades de dicho servicio de modo que sea accesible desde cualquier tipo de cliente a través de protocolos estándar.

El presente documento contiene la investigación base de la Arquitectura de Software Orientada a Servicios (SOA), entregando sus características, ventajas y componentes principales, así como la evolución que da origen a su utilización como plataforma que soporta las Tecnologías de Información dentro de las grandes empresas, antecedentes necesarios para llevar adelante un caso práctico, descrito a partir del capítulo 5.

Este trabajo comienza con una pequeña reseña histórica sobre el concepto y evolución de la arquitectura de software, implícito dentro de la ingeniería de software, para luego continuar con una descripción detallada de SOA. Teniendo claros los conceptos, se plantea un análisis de los objetivos y beneficios que persigue esta arquitectura, tanto del punto de vista empresarial, como tecnológico. En el capítulo 6, se presenta la tecnología *Web Service*, junto a sus características, componentes y funcionamiento.

Por último, se desarrolla un caso práctico que incluye el análisis de la situación actual, el diseño y la implementación de una arquitectura basada en SOA, además de la codificación y descripción de dos servicios específicos que resuelven necesidades comerciales y operativas del caso de estudio. En el caso del primer servicio, Servicio Datos de Contacto, cubre la necesidad comercial de la empresa de mantener la información de contacto (dirección, teléfono, email) consolidad y actualizada en los distintos sistemas operacionales de la empresa, para fines de este caso de estudio se consideran dos sistemas, el primer sistema es uno propietario que contiene toda la información de los productos de pólizas vendidos a los clientes. El segundo sistema es Peoplesoft, en el cual se llevan los flujos de

las campañas publicitarias del Call Center. El objetivo principal de este servicio es el de mejorar la calidad de los datos relacionados al cliente y sus productos para efectuar un marketing más certero.

En el caso del segundo servicio desarrollado en el caso práctico, este se enfoca a un proceso más operacional, donde es necesario mejorar la gestión y el control que se lleva sobre los mandatos de un cliente. Un mandato es un contrato realizado por el cliente donde este acepta el cargo de un pago (de uno o más productos) a su cuenta corriente o a su tarjeta de crédito. El servicio tiene como objetivo principal realizar actualizaciones masivas y una a una de los estados de cada mandato. Los sistemas a integrar son el sistema propietario y Peoplesoft, donde en el primer sistema se registran los datos del mandato, se puede cambiar la vía de pago, ya sea en una propuesta de póliza o en una póliza y en el segundo sistema se establecen los flujos administrativos y de control para validar estados del mandato.

Objetivos del Proyecto

Objetivo General.

Realizar una investigación sobre la Arquitectura Orientada a Servicios (SOA) y su implementación con Web Services, con el fin de aplicar esta tecnología en una Institución Financiera, realizando la integración de información relacionada a una póliza de seguros entre dos sistemas transaccionales.

Objetivos Específicos.

Para el cumplimiento de los objetivos generales, se plantean los siguientes objetivos específicos:

- Investigar los conceptos asociados con la Arquitectura Orientada a Servicios (SOA) que se relacionan con la integración de sistemas.

- Realizar una investigación de la tecnología *Web Services*, para obtener un fundamento teórico que permita implementar la tecnología.

- Efectuar un levantamiento de la arquitectura presente en la empresa definida para realizar un caso práctico.

- Diseñar una solución basada en la metodología de modelado y diseño para implementar una arquitectura SOA (orientadas a servicios), considerando escalabilidad e integración de la arquitectura.

- Desarrollar dos servicios que permitan utilizar la arquitectura planteada. Probando el tiempo de respuesta y la disponibilidad de la arquitectura.

1 Arquitectura Orientada a Servicios

Este capítulo está compuesto por una investigación del concepto de Arquitectura de Software, donde se explica la historia de esta disciplina y los grandes hitos que han permitido su desarrollo a través del tiempo.

Luego el capítulo continúa con una descripción de las opciones en las cuales se puede basar una arquitectura orientada a servicios, considerando una arquitectura orientada a procesos o una arquitectura centrada en datos, estableciendo sus diferencias y características principales. Luego de esto viene una descripción de una Arquitectura Orientada a Servicios (SOA), donde se definen sus componentes y las características de cada una de ellos.

Finaliza el capítulo mencionando cuales son los objetivos de SOA, desde un punto de vista tecnológico y desde un punto de vista empresarial.

1.1 Antecedentes de la Arquitectura de Software.

La Arquitectura de Software remonta sus antecedentes a partir de la década de 1960, siendo una historia discontinua, al contrario de lo que presenta el campo en el que se inscribe, la Ingeniería de Software. Luego de los enunciados de Edsger Dijkstra, de David Parnas y de Fred Brooks, la Arquitectura de Software quedó en un estado latente durante una gran cantidad de años, reactivándose con los manifiestos de Dewayne Perry de AT&T Bell Laboratories de New Jersey y Alexander Wolf de la Universidad de Colorado. Dentro del mundo de la informática se plantea que fueron Perry y Wolf los que fundaron la disciplina, siendo seguidos por gente de la universidad Carnegie Mellon, David Garlan, Mary Shaw, Paul Clements y Robert Allen.

Antes de Perry y Wolf, se formularon las ideas que formaron la base de los planteamientos formulados en la actualidad. En 1968, Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso establecer una estructuración correcta de los sistemas de software antes de comenzar a programar, evitando escribir

código improvisado.[4] Dijkstra, quien sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores, fue uno de los introductores de la noción de sistemas operativos organizados en capas que se comunican sólo con las capas adyacentes y que se superponen entre sí. Ayudó a precisar además, docenas de conceptos; el algoritmo del camino más corto, los stacks, los vectores y los semáforos. De sus ensayos nace la tradición de hacer referencia a “niveles de abstracción”. Aunque Dijkstra no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para lo que luego expresarían Niklaus Wirth [5] como stepwise refinement (refinamiento paso a paso) y DeRemer y Kron [6] como programming-in-the large (o programación en grande), ideas que poco a poco irían decantando entre los ingenieros primero y los arquitectos después.

En 1975, Brooks, diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa [7], empleando una nomenclatura que ya nadie aplica de ese modo. En el mismo texto, identificaba y razonaba sobre las estructuras de alto nivel y reconocía la importancia de las decisiones tomadas a ese nivel de diseño. También distinguía entre arquitectura e implementación; mientras aquella decía qué hacer, la implementación se ocupa de cómo.

Una novedad importante en la década del 70 fue el advenimiento del diseño estructurado y de los primeros modelos explícitos de desarrollo de software. Estos modelos comenzaron a basarse en una estrategia más orgánica, evolutiva y cíclica, dejando atrás las metáforas del desarrollo en cascada que se inspiraban más bien en la línea de montaje de la ingeniería del hardware y la manufactura. Surgieron entonces las primeras investigaciones académicas en materia de diseño de sistemas complejos. Poco a poco el diseño se fue independizando de la implementación, y se forjaron herramientas, técnicas y lenguajes de modelado específicos.

En la misma época, otro precursor importante, David Parnas, demostró que los criterios seleccionados en la descomposición de un sistema impactan en la estructura de los programas y propuso diversos principios de diseño que debían seguirse a fin de obtener una estructura adecuada. Parnas [8] desarrolló temas tales como módulos con ocultamiento de

información, estructuras de software [9] y familias de programas[10], enfatizando siempre la búsqueda de calidad del software, cuantificable en términos de economías en los procesos de desarrollo y mantenimiento.

En 1972, Parnas publicó un ensayo en el que discutía la forma en que la modularidad en el diseño de sistemas podía mejorar la flexibilidad y el control conceptual del sistema, acortando los tiempos de desarrollo [8]. Introdujo entonces el concepto de ocultamiento de información (information hiding), uno de los principios de diseño fundamentales en diseño de software utilizados aún en la actualidad. La herencia de este concepto en la ingeniería y la arquitectura ulterior es inmensa, y se confunde estrechamente con la idea de abstracción.

Decía Parnas que las decisiones tempranas de desarrollo serían las que probablemente permanecerían invariantes en el desarrollo posterior de una solución. Esas “decisiones tempranas” constituyen de hecho lo que hoy se llamarían decisiones arquitectónicas. Como escriben Clements y Northrop [11] en todo el desenvolvimiento posterior de la disciplina permanecería en primer plano esta misma idea: la estructura es primordial (structure matters), y la elección de la estructura correcta debe ser crítica para el éxito del desarrollo de una solución. Ni duda cabe que “la elección de la estructura correcta” sintetiza, como ninguna otra expresión, el programa y la razón de ser de la Arquitectura de Software.

En la década del 80, los métodos de desarrollo estructurado demostraron que no eran suficientemente adaptables y fueron dejando el lugar a un nuevo paradigma, el de la programación orientada a objetos. En teoría, parecía posible modelar el dominio del problema y el de la solución en un lenguaje de implementación. La investigación que llevó a lo que después sería el diseño orientado a objetos puede remontarse incluso a la década del 60 con Simula, un lenguaje de programación de simulaciones, el primero que proporcionaba tipos de datos abstractos y clases, y después fue refinada con el advenimiento de Smalltalk. Paralelamente, hacia fines de la década del 80 y comienzos de la siguiente, la expresión arquitectura de software comienza a aparecer en la literatura para hacer referencia a la configuración morfológica de una aplicación.

Mientras se considera que la ingeniería de software se fundó en 1968, cuando F.L. Bauer usó ese sintagma por primera vez en la conferencia de la OTAN de Garmisch, Alemania, la

AS, como disciplina bien delimitada, es mucho más nueva de lo que generalmente se sospecha. El primer texto que vuelve a reivindicar las abstracciones de alto nivel, reclamando un espacio para esa reflexión y augurando que el uso de esas abstracciones en el proceso de desarrollo puede resultar en "un nivel de arquitectura de software en el diseño" es uno de Mary Shaw [12] seguido por otro llamado Larger scale systems require higher level abstractions. [13]. Se hablaba entonces de un nivel de abstracción en el conjunto; todavía no estaban en su lugar los elementos de juicio que permitieran reclamar la necesidad de una disciplina y una profesión particulares.

El primer estudio en que aparece la expresión "arquitectura de software" en el sentido en que hoy se conoce es el de Perry y Wolf [3], que fue publicado en 1992, aunque el trabajo se había ido gestando desde 1989. En él, los autores proponen concebir la arquitectura de software por analogía con la arquitectura de edificios, una analogía de la que luego algunos abusaron, [14] otros encontraron útil y para unos pocos resultó inaceptable [15]. Un extracto del paper de Perry y Wolf define la disciplina de Arquitectura de Software: "El propósito de este paper es construir el fundamento para la arquitectura de software. Primero desarrollaremos una intuición para la arquitectura de software recurriendo a diversas disciplinas arquitectónicas bien definidas. Sobre la base de esa intuición, presentamos un modelo para la arquitectura de software que consiste en tres componentes: elementos, forma y razón. Los elementos son elementos ya sea de procesamiento, datos o conexión. La forma se define en términos de las propiedades y las relaciones entre los elementos, es decir, las restricciones que operaban sobre ellos. La razón proporciona una base subyacente para la arquitectura en términos de las restricciones del sistema, que derivan frecuentemente de los requerimientos del sistema. Discutimos los componentes del modelo en el contexto de la arquitectura como de los estilos arquitectónicos".

Uno de los acontecimientos arquitectónicos más importantes del año 2000 fue la hoy célebre tesis de Roy Fielding que presentó el modelo REST, el cual establece definitivamente el tema de las tecnologías de Internet y los modelos orientados a servicios y recursos como el centro de las preocupaciones de la disciplina [16]. En el mismo año se publica la versión definitiva de la recomendación IEEE Std 1471, que procura

homogeneizar y ordenar la nomenclatura de descripción arquitectónica y homologa los estilos como un modelo fundamental de representación conceptual.

1.2 Tipos de Arquitectura Orientada a Servicios.

En el presente apartado se presentarán algunas de las mejores opciones para las arquitecturas orientadas a servicios. Estas opciones sirven como fundamento a las actuales opciones arquitectónicas que puedan ser agregadas. Las dos opciones presentadas corresponden a arquitectura centrada en datos y de proceso distribuidos.

1.2.1 SOA Centrada en datos.

Este tipo de arquitectura se basa en el movimiento de los datos hacia donde puedan ser necesarios. Esto se hace con un *router* de mensaje y los datos que son solicitados con mayor frecuencia por varios servicios. Las principales ventajas de una arquitectura centrada en los datos incluyen:

La calidad de los datos es controlada. Existe una copia principal de cualquier ítem de datos, donde se refleja el control de calidad, y cada nuevo ítem de datos es un duplicado de la copia principal. El *router* del mensaje controla la redundancia de datos en el diseño de la aplicación.

El efecto sobre los sistemas operacionales se reduce al mínimo. Esta premisa es efectiva para sistemas existentes y nuevos. El impacto de una arquitectura centrada en datos ocurre en el momento de realizar la actualización de los datos desde el *router* de mensaje, momento en el cual debe mantenerse en espera, logrando con ello sufrir un mínimo efecto.

Pocos componentes necesitan tener alta disponibilidad. Generalmente, solo la base de datos central (master) y el *router* de mensaje necesitar estar altamente disponibles en una arquitectura centrada en datos. Sin embargo, las necesidades específicas de una organización pueden dictar que otros sistemas y servicios se mantengan en alta disponibilidad.

Las desventajas de una arquitectura centrada en datos incluyen:

Retraso en obtener actualizaciones de los datos distribuidas. Esto ocurre porque en el traspaso de datos mediante el *router*, puede haber demora. En ciertos casos las actualizaciones inmediatas son necesarias, siendo una de las mayores debilidades ya que este tipo de arquitectura no soporta esta función.

Decidir qué datos se deben enviar. Esto es una decisión de diseño. La desventaja es que los datos requieren ser diseccionados en etapas relativamente tempranas del diseño de una arquitectura centrada en datos. Por supuesto, los datos adicionales pueden ser encaminados siempre. Sin embargo, es algo que generalmente necesita ser definido con anterioridad por el diseñador o arquitecto de software para el desarrollo de la arquitectura.

1.2.2 SOA de Procesos Distribuidos.

Una alternativa a una arquitectura centrada en datos es una arquitectura de procesos distribuidos. Se le llama distribuido porque el proceso ocurre en múltiples ubicaciones. Estas ubicaciones pueden corresponde a cualquier localización donde el sistema tenga requerimiento de datos o de procesos.

Se debe considerar una arquitectura de procesos distribuidos en aquellas situaciones en las cuales es crítico tener disponible online tipos de procesamiento o datos. Por ejemplo, en una empresa de viajes sería importante tener la información actualizada sobre los autos arrendados junto con las reservaciones de una línea aérea y del hotel. En otros casos, pudieran ser necesarios los últimos datos sobre los precios comunes de un cierto tipo de producto. En todos estos escenarios, si los datos necesitan estar altamente disponibles, se necesita desplegar la arquitectura de procesos distribuidos en un hardware y software altamente disponibles.

1.2.3 Comparación entre proceso y centrada en datos.

Para comparar los dos acercamientos, se puede observar una petición simple que debe obtener toda la información del cliente. En la Figura 1-1 se muestra esta petición usando una arquitectura centrada en datos. Los datos provienen a partir de una fuente: el archivo principal de cliente. Esto corresponde a un servicio altamente disponible. La petición no afecta otros sistemas internos.

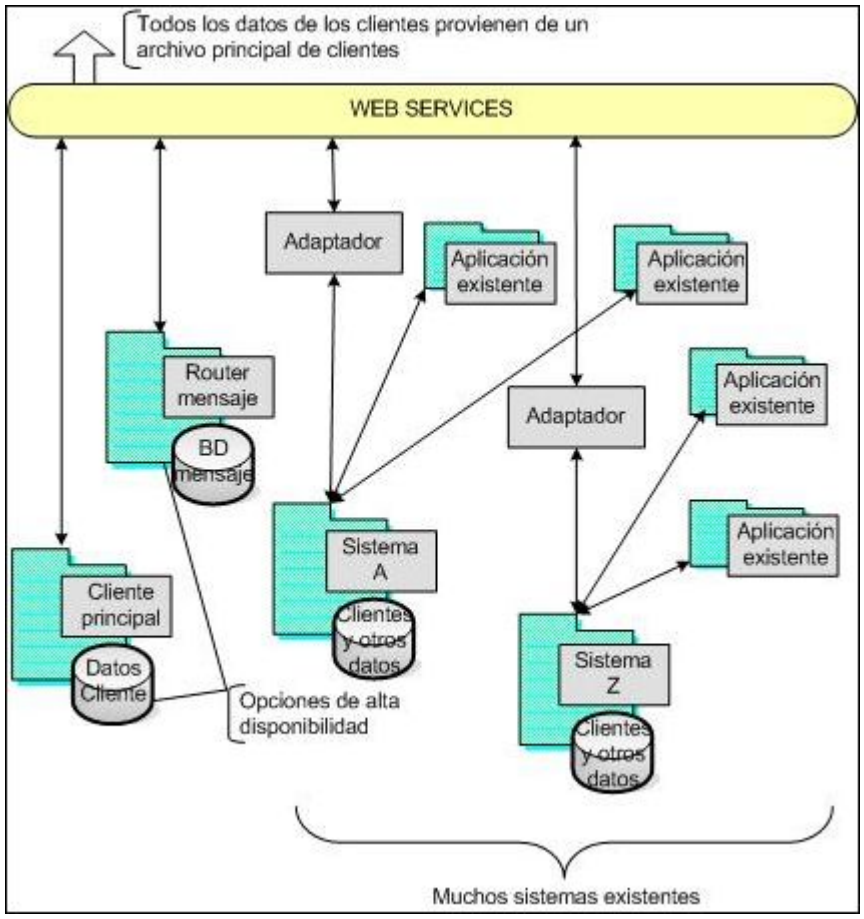

Figura 1-1: Arquitectura centrada en datos, una respuesta a una petición de todos los datos del cliente a partir de una fuente de alta disponibilidad.

La Figura 1-1 muestra la misma petición sin un archivo principal de cliente. Los datos provendrían de localizaciones múltiples. El solicitante se ocuparía de los duplicados y de las posibles inconsistencias en los datos. En ese caso, si ninguno de los sistemas internos estuviera disponible, no se entregaría toda la información del cliente. Finalmente, las solicitudes de este tipo son la perdición de los administradores de operación de sistemas, debido a que pueden retrasar los sistemas operacionales y, a menudo, ocurrir en horas imprevistas.

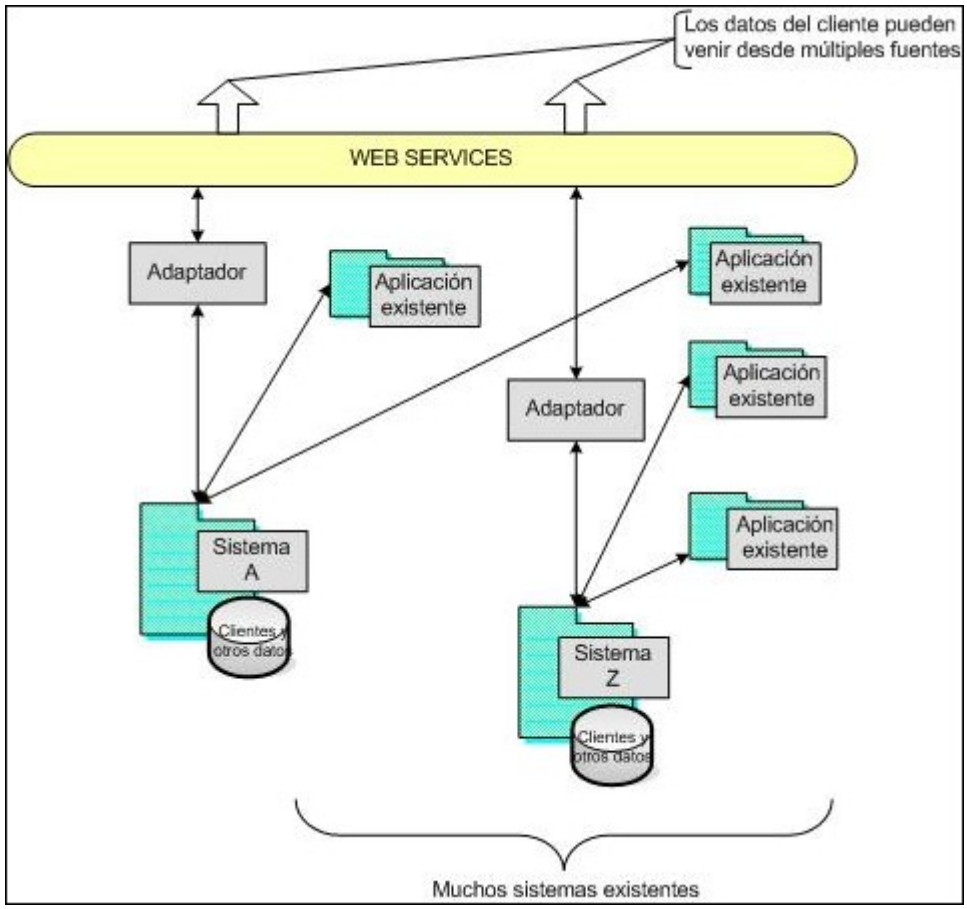

Figura 1-2: Arquitectura de procesos distribuida, una respuesta a una petición de todos los datos del proveniente de fuentes múltiples.

Para ocuparse de la disponibilidad de datos, cada uno de los sistemas internos se podría hacer altamente disponible como el sistema de la Figura 1-2 en ese caso, cada sistema necesitaría estar altamente disponible para igualar la disponibilidad del servicio enfocado al cliente. Obviamente para una arquitectura de proceso distribuida sería considerablemente más costoso proporcionar una respuesta altamente disponible y completa que una arquitectura centrada en datos. Hay, sin embargo, razones muy buenas para considerar una arquitectura de procesos distribuidos cuando no es necesario preocuparse de la redundancia y de la consistencia de datos.

1.2.4 Combinando arquitecturas centradas en datos y de procesos distribuidos.

En la práctica, la mayoría de las arquitecturas orientada a servicios llegarán a combinar las arquitecturas centradas en datos con la de procesos distribuidas. Por ejemplo, cuando un servicio de agencia de viajes necesita la información actualizada de las reservaciones. Para poder competir en los negocios, los servicios tales como arriendo de autos y reservaciones de líneas aéreas deben proporcionar estos datos en el momento. La base de datos principal podía tener una entrada de datos o actualización con un leve retraso, sin impactar en la organización.

1.3 Descripción de Arquitectura Orientada a Servicios.

Una arquitectura del software es un conjunto de parámetros que definen y asignan las funcionalidades del sistema a los componentes de software. Así mismo, describen la estructura, restricciones, y características técnicas de los componentes y de las interfaces que los comunican. La arquitectura es el modelo del sistema y por lo tanto el plano implícito de alto nivel para su construcción.

Una arquitectura SOA se basa en cuatro abstracciones claves: aplicación cliente, servicio, directorios del servicio y un bus de servicio, representados en la Figura 1-3. [33] A pesar de que la aplicación cliente es la componente propietaria de los procesos de negocio, los servicios proporcionan la funcionalidad del negocio que la aplicación cliente y otros servicios pueden utilizar. Un servicio consiste en una implementación que proporciona la lógica y los datos del negocio, una descripción que especifica la funcionalidad, uso y las restricciones para los clientes de un servicio, además de una interfaz que expone físicamente la funcionalidad. El directorio del servicio almacena la descripción de los servicios individuales de un SOA, y el bus del servicio interconecta las fronteras y los servicios.

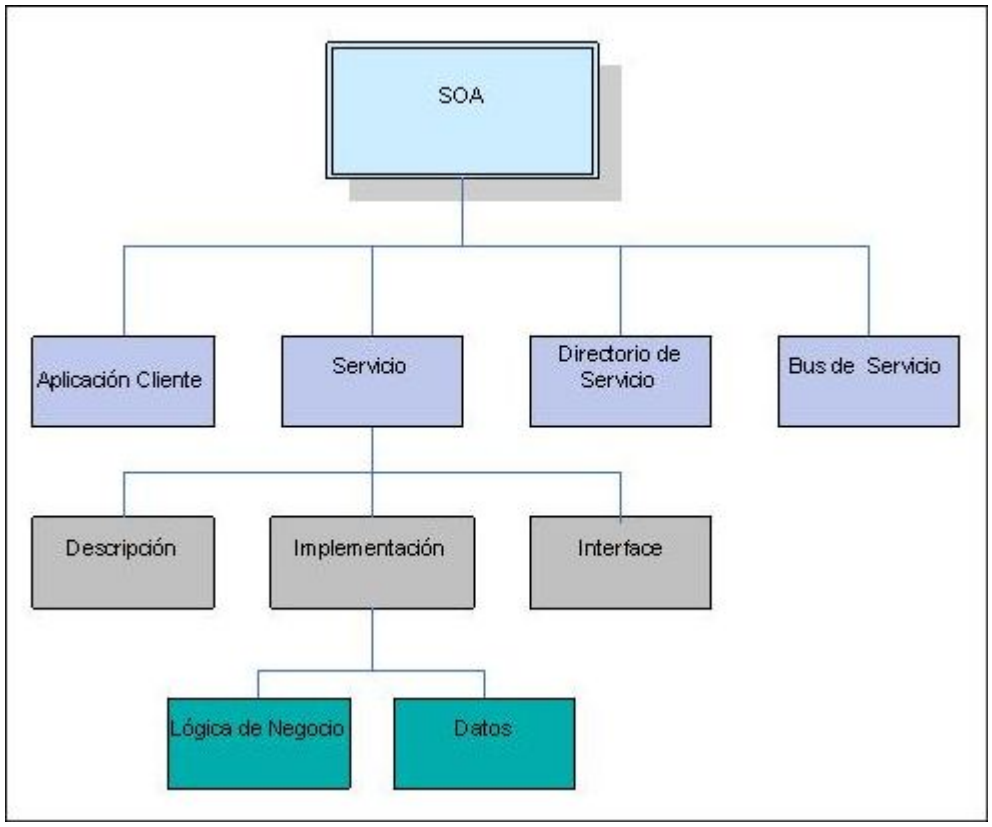

Figura 1-3 Componentes principales de una Arquitectura SOA.

El concepto de SOA se centra en la definición de una infraestructura del negocio. Al utilizar el término “servicio,” se puede hablar de reservaciones de una línea aérea o el acceso a la base de datos del cliente de una compañía. Estos servicios proporcionan operaciones de negocio, como conseguir una reservación, cancelan la reservación, o conseguir el perfil de un cliente. Así como los servicios de negocio, existe la infraestructura de servicios técnicos, que se encarga, por ejemplo, de comenzar una transacción, actualizar datos o abrir un cursor. Si bien este tipo de funcionalidad es muy útil al ejecutar una operación de negocio, tiene poca importancia estratégica desde el punto de vista de SOA. En general, la tecnología no debe tener ningún impacto en la estructura de alto nivel de la solución o causar dependencias entre los componentes. Esto significa que SOA debe desacoplar las aplicaciones de negocio de los servicios técnicos, y hacer a una empresa independiente de una infraestructura técnica específica.

Las aplicaciones cliente son elementos activos del SOA, que entregan el valor del SOA a los usuarios finales. Sin embargo, se debe tener siempre presente que los servicios proporcionan la estructura al SOA. Si bien los servicios pueden permanecer estables, las aplicaciones deben estar en continua adaptación a cambios, al igual que los procesos del negocio de las empresas. Por lo tanto, el ciclo de vida de las aplicaciones cliente es mucho más corto que el ciclo de vida de los servicios subyacentes. Esta es la razón por la cual los servicios son considerados como las entidades de mayor importancia estratégica en un SOA.

1.3.1 Aplicaciones Cliente.

Las aplicaciones cliente inician y controlan todas las actividades de los sistemas de una empresa. Hay diversos tipos de aplicaciones cliente. El ejemplo más común corresponde a una interfaz de usuario gráfica, tal como un Web browser o un cliente que interacciona directamente con los usuarios finales. Sin embargo, las aplicaciones cliente no necesariamente tienen que interactuar en forma directa con los usuarios finales. Los programas *batch* o los procesos que invocan periódicamente funcionalidades también son ejemplos válidos de aplicaciones cliente.

Sin embargo, es muy posible que una aplicación cliente delegue mucha de su responsabilidad sobre un proceso del negocio a unos o más servicios. En última instancia, sin embargo, es siempre una aplicación cliente la que inicia un proceso de negocio y recibe los resultados.

1.3.2 Servicios.

Como concepto, un servicio corresponde a una ubicación en la red que tiene una descripción legible por la máquina que recibe y que retorna los mensajes opcionalmente. Un servicio por lo tanto se define en términos de patrones de intercambio de mensajes que soporta. Se utiliza un esquema para los datos contenidos en el mensaje como parte principal del contrato establecido entre un solicitante y un proveedor de servicio. Otros ítems de meta datos describen la dirección de red para el servicio, las operaciones que soportan, y sus requisitos de confiabilidad y seguridad [32].

En términos reales, un servicio es un componente de software que en general encapsula un concepto de alto nivel del negocio. Se pueden identificar las siguientes partes:

- **Contrato**. El contrato de servicio proporciona una especificación informal del propósito, funcionalidad, restricciones, y del uso del servicio. El formato de esta especificación puede variar, dependiendo del tipo de servicio. Puede contener una definición de interfaz formal basada en lenguajes como IDL o WSDL. Aunque no es obligatorio, una definición formal de interfaz de servicio agrega una ventaja significativa: Proporciona abstracción e independencia adicional de la tecnología, incluyendo el lenguaje de programación, *middleware*, protocolo de red, y el variables de ambiente. Sin embargo, es importante entender que el contrato de servicio proporciona más información que una especificación formal. El contrato puede proveer una semántica detallada de la funcionalidad y parámetros que no están sujetos a especificaciones de IDL o de WSDL. En realidad, muchos proyectos deben hacer frente a servicios que no pueden proporcionar descripciones formales del interfaz del servicio.

En estos casos, el servicio puede entregar bibliotecas de acceso o una descripción técnica detallada a nivel de protocolo de red. Sin embargo, cada servicio requiere un contrato si no tiene disponible una descripción formal basada en un estándar tal como WSDL o IDL.

- **Interfaz**. La funcionalidad del servicio se entrega a través de la interfaz de servicio a los clientes que están conectados utilizando una red. Aunque la descripción de la interfaz es parte del contrato de servicio, su implementación física consiste en trozos del servicio, los cuales son incorporados a los clientes de un servicio y ejecutados.

- **Implementación**. La implementación física del servicio provee la lógica del negocio que se requiere y los datos apropiados. Es la realización técnica que cumple el contrato de servicio. La implementación del servicio consiste en unos o más componentes como programas, datos de configuración, y bases de datos.

- **Lógica del negocio.** La lógica del negocio encapsulada por un servicio es parte de su implementación. Provee la disponibilidad directa de las interfaces del servicio. Sin embargo, se recomienda la programación no orientada a interfaces, con el fin de facilitar la orientación a los servicios.

- **Datos**. Un servicio también puede incluir datos. Específicamente es el propósito de un servicio orientado al dato.

Los servicios no son solo la encapsulación de un cierto código en las capas más bajas de una aplicación. Cada servicio es una entidad funcional especializada que encapsula una entidad de negocio de alto nivel. Por lo tanto, desde la perspectiva del cliente, un servicio es una entidad que funciona como una caja negra.

1.3.3 Directorio de Servicio.

Un directorio de servicio proporciona la infraestructura para identificar los servicios y obtener la información para utilizarlos, particularmente si estos servicios se deben obtener fuera del alcance funcional y temporal del proyecto que los creó. Si bien mucha de la información requerida forma parte del contrato de servicio, el directorio puede proporcionar información adicional, como ubicación física, información sobre el proveedor, contacto con personas, cobros por uso, restricciones técnicas, temas de seguridad, y niveles de disponibilidad de servicio.

Estas características se aplican a los directorios de servicio que se utilizan dentro de los límites de una sola empresa. Los directorios que se utilizan para la integración de servicios entre varias empresas, generalmente tienen distintos requisitos si los requerimientos exigen que se hagan públicos a través de Internet. Estos requisitos pueden abarcar temas legales (términos y condiciones de uso), estilos de presentación, seguridad, registro del usuario, suscripción del servicio, facturación, y versiones.

Obviamente, un directorio de servicio es un elemento muy útil en SOA. Aunque se puede construir un SOA y alcanzar muchas de sus ventajas sin establecer un directorio de servicio, es imprescindible a largo plazo. Una arquitectura puede hacer frente sin un directorio si el alcance de un servicio es solo un proyecto, si tiene muy pocos servicios, o si todos los proyectos se conforman por los mismos miembros de un equipo. En general, la mayoría de los escenarios de una empresa se caracterizan por el desarrollo de muchos proyectos concurrentes, equipos que cambian, y una variedad de servicios.

Un directorio de servicio puede ser arbitrariamente simple y no requerir ninguna tecnología. Un lote de contratos de servicio impresos situados en una oficina y disponibles para todos los proyectos es un directorio válido del servicio. Sin embargo, existen mejores maneras para proporcionar esta información conservando la simplicidad del directorio. A menudo, se puede encontrar un tipo de base de datos propietaria que contiene ciertos datos administrativos formales y el contrato de servicio más o menos formal para cada versión de un servicio.

En algunos casos, las compañías han desarrollado herramientas propias que generan automáticamente la descripción y las definiciones formales del servicio (por ejemplo, un generador del HTML que toma WSDL como entrada, similares a un generador de JavaDoc). Los siguientes son ejemplos de la información que deben contener un directorio de servicio de una empresa:

Servicio, operación y firmas de los acuerdos, por ejemplo, el formato de definiciones del esquema WSDL y de XML.

Propietario del servicio. En una empresa SOA, los propietarios pueden operar en el nivel de negocio (responsables de las consultas y cambiar requerimientos a nivel funcional), nivel técnico (responsables de consultas técnicas y de solicitudes de cambio), y nivel operativo (responsables de consultas con respecto a las mejores maneras de accesar un servicio, o de problemas operacionales).

Derechos de acceso, como información sobre las listas del control de acceso y el mecanismo de seguridad, o una descripción del proceso que se debe seguir dentro de la empresa para que un nuevo sistema pueda utilizar un determinado servicio.

Información sobre el funcionamiento y la performance previstos del servicio, incluyendo los tiempos medios de reacción, y limitaciones potenciales del rendimiento de procesamiento. Esto se puede resumir como parte de una plantilla genérica de SLA (acuerdo del nivel de disponibilidad).

Características transaccionales del servicio y de sus operaciones individuales. Esto incluye la información sobre las características de lectura/escritura de actualización, si la operación es idempotente, y la lógica de remuneración asociada

1.3.4 Bus de Servicio.

Un bus del servicio conecta a todos los participantes de una SOA y cada una de las aplicaciones cliente. Si dos participantes necesitan comunicarse, por ejemplo, si una aplicación cliente necesita invocar una cierta funcionalidad de un servicio básico el bus de servicio lo lleva acabo. El bus de servicio no se compone necesariamente de una sola tecnología, sino que abarca una gran variedad de productos y de conceptos. Las características principales son las siguientes:

Conectividad. El principal propósito del bus de servicio es interconectar los componentes de una SOA. Proporciona la infraestructura que permite a las aplicaciones cliente y los servicios invocar la funcionalidad de servicios.

Heterogeneidad de la tecnología. El bus de servicio debe abarcar una variedad de diversas tecnologías. La realidad de las empresas se caracteriza por utilizar tecnologías heterogéneas. Por lo tanto, el bus de servicio debe poder conectar a los participantes que se basan en diversos lenguajes de programación, sistemas operativos, o variables de entorno. Además, generalmente en las empresas utilizan una multiplicidad de productos de middleware y protocolos de comunicación, y todo el esto se debe apoyar por el bus de servicio.

Heterogeneidad de los conceptos de comunicación. Similar a la heterogeneidad de tecnologías, el bus de servicio también debe abarcar una variedad de diversos conceptos de comunicación. Debido a la divergencia de los requisitos de diversas aplicaciones, el bus de servicio debe permitir diferentes modos de comunicación, y como mínimo, instalaciones para la comunicación síncrona y asincrónica.

Servicios técnicos. Aunque el propósito del bus de servicio es sobre todo comunicación, debe proporcionar también servicios técnicos tales como registro, revisión, seguridad, transformación del mensaje, o transacciones.

1.4 Objetivos de SOA.

Existen dos enfoques que permiten visualizar los objetivos de implantar una SOA sobre una organización. El primer enfoque da un punto de vista empresarial, justificando la implantación a través de los beneficios que la empresa obtendrá. El segundo punto de vista es tecnológico, donde se destaca el impulso que la organización obtendrá al implantar nuevas tecnologías.

1.4.1 Punto de Vista Empresarial.

Cuando una empresa decide hacer uso de una arquitectura SOA [17] es porque tiene objetivos específicos de negocio que cubrir, reducir costes, aumentar ingresos, mejorar la productividad, comunicación inter-empresarial con varias empresas y ajustar los sistemas a los requerimientos del negocio.

También se puede decir que una arquitectura SOA consiste en una forma de modularizar los sistemas y aplicaciones en componentes de negocio que pueden combinarse y recombinarse con interfaces bien definidas para responder a las necesidades de la empresa.

Con el uso de entornos orientados a servicios las empresas pretenden mejorar la interacción con los clientes, *partners*, proveedores, empleados y también reducir el ROI (*Return of Investment*) retorno de la inversión, es decir, conseguir una mayor rentabilidad de las inversiones tecnológicas.

Para las empresas se abre un abanico amplio de aplicación, desde la utilización en la cadena de suministro, entornos B2B o servicios de seguridad

Como se ilustra en la Figura 1-4, la tendencia de muchas empresas converge a manejar una TI principal, cambiando los conceptos de implementación hacia una orientación a servicios. Las tecnologías claves en esta convergencia son [32]:

Administración de procesos de negocio (BMP) metodologías y tecnologías para automatizar operaciones de negocio cuyos objetivos se orientan a :

Describir explícitamente los procesos del negocio de modo que sean más fáciles de entender, refinar, y optimizar.

Facilitar la rápida modificación de procesos de negocio según los cambios en sus requisitos.

Automatizar los procesos manuales y hacer cumplir las reglas de negocio.

Proporcionar la información y el análisis en tiempo real de los procesos del negocio para apoyar a los responsables.

Web Services XML basado en tecnología de mensajería, descripción de servicios, y características extendidas, provee:

Estándares abiertos para las descripciones distribuidas de interfaces computacionales e intercambio del documento vía mensajes.

Independencia de las plataformas subyacentes a la ejecución y el uso de la tecnología.

Aumento de las cualidades del servicio de las empresas tales como seguridad, confiabilidad, y transacciones.

Soporte para las aplicaciones complejas que trabajan con flujo de procesos, accesos multi-canal, e integración rápida entre otros.

Servicios Orientados a la Arquitectura (SOA): metodología para alcanzar la interoperabilidad en el uso y la reutilización de los activos TI que ofrece:

Una fuerte base arquitectónica, incluyendo control, procesos, modelado, y herramientas.

Un nivel óptimo de abstracción para alinean las necesidades del negocio y las capacidades técnicas, creaciones reutilizables, funcionalidad del *core* del negocio.

Una infraestructura de desarrollo en la que nuevas funcionalidades puede ser construidas rápida y fácilmente.

Una biblioteca de servicios reutilizable para funciones de negocio y TI similares.

XML (*Extensible Markup Language*): estándar que provee un manejo de formato para los datos de manera independiente de como los genera o provee una empresa, además de:

Tipos de datos y estructuras estándar, independientes de cualquier lenguaje de programación, ambiente de desarrollo, o sistema de software.

La tecnología requerida para definición del negocio y la creación de documentación e información que evoluciona, incluyendo vocabulario estándar para muchas industrias.

El software adecuado para manejar operaciones sobre XML, incluyendo parsers, queries, y transformaciones.

Individualmente, cada una de estas tecnologías tiene un efecto profundo en unos o más aspectos del negocio tecnológico. Cuando están combinados, proporcionan una plataforma cooperativa para obtener las ventajas de la orientación al servicio y avanzar al próximo paso en la evolución de los sistemas TI.

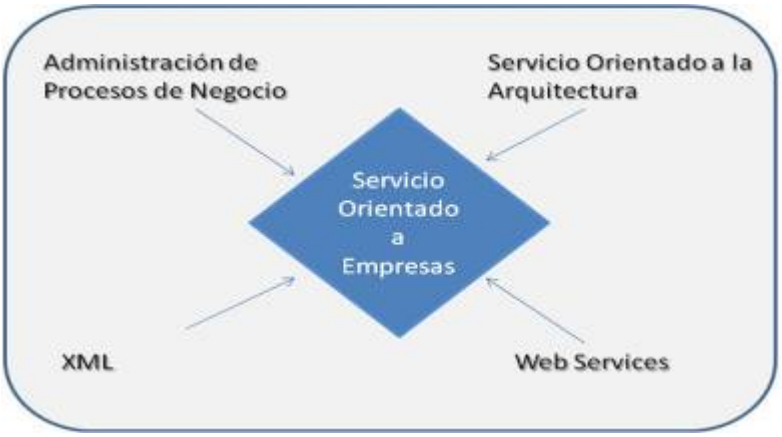

Figura 1-4: Tendencia Orientación a Empresas.

Beneficios para el negocio.

- **Eficiencia**. Transforma los procesos de negocio en servicios compartidos con un menor coste de mantenimiento.

- **Capacidad de respuesta**. Rápida adaptación y despliegue de servicios, clave para responder a las demandas de clientes, partners y empleados.

- **Adaptabilidad**. Facilita la adopción de cambios añadiendo flexibilidad y reduciendo el esfuerzo.

1.4.2 Punto de Vista Tecnológico.

Las arquitecturas SOA pretenden concebir las aplicaciones desde otro punto de vista, una aplicación orientada a servicios combina datos en tiempo real con otros sistemas capaces de fusionar los procesos de negocio.

Las aplicaciones basadas en SOA utilizan tecnología totalmente estándar como es XML y Web Services para la mensajería. Estándares como SOAP [18], Web Services Description Language (WSDL) [19] y Business Process Execution Language (BPEL) [20], estandarizan el traspaso de información, el modelo de integración de procesos y la cooperación entre aplicaciones.

Desarrollar un servicio es diferente a desarrollar un objeto, porque un servicio es definido por los mensajes que intercambia con otros servicios. Un servicio se debe definir con un mayor nivel de abstracción (como decir el denominador común más bajo) que un objeto, porque es posible mapear la definición del servicio a un procedimiento orientado al lenguaje como COBOL o PL/I, o a un sistema que utiliza mensajes como JMS o MSMQ, así como a sistemas orientados al objeto como J2EE o Framework.NET.

Realizando aplicaciones orientadas a servicio se pueden conectar aplicaciones heterogéneas con el aumento de flexibilidad que supone, y un punto muy importante es que permite que las organizaciones interactúen cuando realmente lo requieran, sin necesidad de tener conexiones permanentes.

También es importante entender la granularidad en que debe ser definido el servicio. Un servicio define normalmente una interfaz de grano grueso (multiprocesamiento de procesos

concurrentes en un entorno multiprogramado) que acepte más datos en una sola invocación que un objeto y consuma más recursos que un objeto, debido a la necesidad de levantar un ambiente de ejecución y procesar el XML, disponiendo a menudo de accesos remotos. Por supuesto, los interfaces del objeto pueden ser de grano grueso. El punto es que los servicios están diseñados para solucionar problemas de interoperabilidad entre las aplicaciones y para usar los componentes de nuevas aplicaciones o sistemas, pero no crear la lógica detallada del negocio para las aplicaciones.

Los servicios son ejecutados intercambiando mensajes según uno o más patrones que apoyan el intercambio del mensaje (MEPs), por ejemplo la petición/respuesta, asincrónico unidireccional, o publicar/suscribir.

En un nivel del proyecto, el arquitecto generalmente supervisa el desarrollo reutilizable de servicios e identifica medios de almacenamiento, administración, y descripciones de recuperación de servicios cuando y donde sean necesarias. La capa de servicios reutilizables encapsula las operaciones de negocio, por ejemplo, “obtiene un cliente” o “agrega una orden” desde las distintas plataformas de software, de igual forma como los Web Services aíslan el World Wide Web de la variaciones de sistemas operativos y lenguajes de programación. La capacidad de reutilizar servicios provee aplicaciones que crecen rápida y fácilmente, proporcionando a la organización las ventajas de automatizar procesos y de la agilidad para responder a las condiciones cambiantes.

Una forma a ayudar a desarrollar la orientación hacia el diseño, implementación, y utilización de aplicaciones usando servicios, puede ser dividir la responsabilidad dentro de los departamentos TI de la siguiente manera:

Crear el tráfico de servicios responsables de la complejidad de la tecnología subyacente en la cual el servicio se despliega y se asegura que las descripciones XML/servicios Web corresponden a los requerimientos del consumidor del servicio, y que los datos sean correctamente compartidos.

Consumir los servicios que levantan nuevos usos y flujos compuestos del proceso de negocio, asegurándose de que los datos compartidos y los flujos de proceso reflejan exactamente los requisitos operacionales y estratégicos del negocio.

Esta división potencial de la responsabilidad separa con mayor claridad los temas técnicos de los de negocio.

Beneficios Tecnológicos.

Reduce la complejidad gracias a la compatibilidad basada en estándares frente a la integración punto a punto. Reutiliza los servicios compartidos que han sido desplegados previamente. Integra aplicaciones heredadas limitando así el coste de mantenimiento e integración. Facilita el desarrollo, ya que las aplicaciones son reutilizables, más fácil de mantener y tienen la capacidad de ampliación de las funcionalidades del sistema, exponiéndolas de una forma segura.

Resumen del Capítulo.

La Arquitectura de Software es una disciplina relativamente nueva en cuanto a su concepto actual, sin embargo esta ha sido forjada desde los primeros sistemas de información creados, estableciendo distintos conceptos los cuales han ido evolucionando con el tiempo.

El rol de la disciplina, actualmente, está dado por establecer un orden que permita facilitar la integración de distintos sistemas, comunicación entre sistemas, apoyo al proceso de cada corporación, mantención de proyectos actuales y desarrollo de nuevos proyectos. El concepto de SOA, permite establecer este orden, dando un patrón capaz de facilitar desarrollos, aumentar funcionalidades reutilizables y facilitando la integración entre distintas plataformas.

A un nivel macro los conceptos de SOA son cuatro (aplicaciones clientes, servicios, directorio de servicios y bus de servicios), las aplicaciones clientes son todas aquellas que requieren para su funcionamiento consumir un servicio, desplegando los resultados al usuario final, un ejemplo es un portal de pagos que realiza la consulta de los pagos para un

cierto servicio y un cierto cliente, en este caso la interfaz es estándar para todos los portales de pago y lo que se despliega es una respuesta estándar. Los servicios, son las funciones o rutinas que ante una entrada estándar retornan una salida estándar, básicamente es donde reside la lógica de negocios.

En el caso del bus de servicio es el canal de comunicación por donde se realiza la interacción entre los clientes y los servicios. El directorio de servicios es donde se publican los servicios, en este caso se despliega una lista de los servicios disponibles y su descripción.

2 Principios de Diseño SOA

El presente capítulo presenta los principios de diseño de servicios Web que se pueden utilizar para apoyar una arquitectura orientada a servicio (SOA). Entre estos temas se encuentra la definición de un servicio de negocio, y su orientación al contexto de SOA. Posteriormente se presenta un análisis del diseño utilizando WSDL y los medios formales para definir una interfaz del servicio usando XML y WSDL. A partir de este punto, se plantea patrones de diseño desde el punto de vista de e-bussiness para problemas tecnológicos. Finaliza el capitulo con una revisión del soporte tecnológico que existe en las plataformas de desarrollo J2EE y .NET.

2.1 Definir un Servicio de Negocio.

La tarea de definir un servicio de negocio cobra importancia debido a que las compañías requieren poner las aplicaciones a disposición de usuarios internos y externos, resultando un aspecto crítico el que las interfaces expuestas estén bien definidas, sean fáciles de utilizar y estén al servicio del consumidor. Esto significa que deben reflejar eficazmente los conceptos del negocio y los requerimientos (documentos, procesos) mas que los conceptos técnicos (APIs, tipos de datos y plataformas). Las interfaces expuestas de esta manera son referenciadas como servicios de negocio, y con esta lógica, se consigue una aproximación al concepto de Arquitectura Orientada a Servicios (SOA). Al diseñar servicios de negocios se deben considerar los siguientes factores:

Granularidad Una pregunta recurrente en relación a la construcción de sistemas distribuidos es ¿cuál es el tamaño adecuado para una interfaz o granularidad para una entidad dentro del sistema?. En los modelos anteriores, corresponde a la lectura de las entradas de los objetos, componentes o las clases. En SOA, son servicios de negocio usados dentro del contexto de un proceso de negocio que necesitan ser entendidos por los usuarios del negocio. Por lo tanto, una interfaz de servicio de negocio generalmente debe reflejar documentos de negocio del mundo real: formatos, órdenes, contratos, y facturas. En este

contexto, se presenta una granularidad mucho mas gruesa que en los modelos anteriores de integración que se centraron en APIs de programación de bajo nivel.

Reutilización En muchos casos, la reutilización puede ser la mejor estrategia para modelar los servicios basados en formatos de documentos existentes. Para servicios de aplicaciones financieras se puede materializar en un movimiento de dinero SWIFT MT100, o en una aplicación de empresas podía ser una orden SAP R/3. Si bien éstos no pueden ser documentos de SOAP o aún de XML, representan históricamente un mensaje o un documento dentro del contexto de un proceso de negocio. Ahora bien, tecnologías como *Cape Clear Data Interchange* apoyan la transferencia y transformación de datos en esos formatos para SOAP. Por lo tanto, existen procesos de negocio que se pueden reutilizar e integrar en un modelo de SOA. Esta lógica, que permite que un servicio sea invocado por una gama de documentos incluyendo formatos heredados, se conoce como *loose coupling* (acoplamiento ligero).

La interfaz no es igual a la implementación. La opción de definir la cara visible del servicio de negocio no excluye la reutilización de componentes de grano fino (paralelismo inherente en un único flujo de instrucciones), basado en un modelo Java/.Net/CORBA, dentro de la implementación física del servicio. Sin embargo, debe ser transparente para el cliente que utiliza el servicio (es decir la definición del servicio debe evitar referencias de bajo nivel a los objetos utilizados en la implementación del servicio).

Estado. Un documento intercambiado entre servicios de negocio debe ser autónomo, y establecer su contexto en términos de identidad y estado dentro de un proceso de negocio. Los documentos dentro de un SOA no deben confiar en la administración o referencias de componentes externas. Esto es se aplica preferentemente en los modelos síncronos (HTTP) y asincrónicos (JMS). En el primero, el balanceo de carga a nivel HTTP se usa con frecuencia para escalar el estado de la aplicación, por lo tanto se mantiene en la base de datos mas que en la capa de servicio Web. En el caso de JMS, las transacciones podrían tomar horas o aún días con los documentos que sobrevivían a ciclos de procesos del sistema. Por este motivo, es recomendable evitar las referencias de implementación dentro de la estructura del documento.

2.2 Considerar el diseño WSDL.

Una de las contradicciones al principio de los servicios de negocio es que los diseñadores necesitan poder definir la interfaz del servicio sin referenciar a la API técnica existente. Esto funciona contrariamente a muchas implementaciones de plataformas de servicio Web en las que se utiliza una API (interfaz de Java/EJB) como punto de partida para la implementación, donde el diseñador debe "primero codificar" para ordenar y obtener el servicio. Si bien es atractivo para algunas aplicaciones, existen muchas desventajas a este enfoque, incluso para el caso de simples usos del estilo RPC:

WSDL (*Web Services Description lenguaje*) creado desde un código fuente es menos costoso que crearlo desde el esquema original de XML. Hay una gran cantidad de caracteres que no son soportados por los generadores automáticos de WSDL (numeraciones, rangos de validación, concordancia con el modelo, reglas de cadinalidad). Por lo tanto, muchos de los valores de un XML respecto a definiciones de datos se pierden.

La creación de servicios Web implementados desde código fuente puede conducir a errores de interoperabilidad entre plataformas. Por ejemplo, un servicio que utiliza referencias remotas de Java no es interoperable con un cliente basado en .NET. .NET simplemente no comprende que debe hacer con una referencia remota de Java. Usando una metodología de diseño de servicios Web se pueden evitar los casos de interoperabilidad, puesto que el cliente y el servidor están trabajando sobre un sistema común que utiliza XML.

Utilizar un diseño de WSDL, comenzando con un esquema XML y su importación en el WSDL para construir la especificación final del servicio, también trae las siguientes ventajas:

Al crear WSDL primero entonces se puede utilizar herramientas automatizadas para generar el código del cliente y del servidor que pone el servicio en ejecución. Esto conduce a una mejora considerable en la productividad puesto que reduce en forma masiva la cantidad de código que se debe escribir. Esto es aplicable a interfaces complejas donde es

difícil “codificar primero”. Generar un código de interfaz para cada tipo de dato dentro de un desarrollo complejo ahorra una cantidad considerable de trabajo.

Separación del diseño y el desarrollo. Los diseñadores de servicios pueden especificar requisitos como WSDL y entregárselos a los desarrolladores para su implementación.

Los proveedores y consumidores de servicios pueden trabajar en paralelo. Para definir una interfaz de servicio como WSDL, los desarrolladores de la aplicación cliente pueden trabajar independientemente de los desarrolladores de la aplicación servidor. El cliente y el servidor no necesitan utilizar el mismo lenguaje de programación, permitiendo aumentar significativamente el número de personas que pueden participar en el desarrollo de la aplicación distribuida.

WSDL es portable a través de los *frameworks* del servicio Web, haciendo posible independizar con eficacia el diseño del servicio y su implementación.

2.3 Soporte de J2EE para SOA

La plataforma Java 2 edición empresarial (J2EE), es una de las dos principales plataformas utilizadas actualmente para desarrollar soluciones empresariales utilizando servicios Web. En este aparatado se presentarán los componentes principales de la plataforma J2EE para implementar SOA.

2.3.1 Descripción de la Plataforma.

La plataforma Java 2 es un ambiente de desarrollo y compilación basado en el lenguaje de programación Java. Es una plataforma estandardizada que soporta a los desarrolladores proporcionando herramientas de programación, desarrollo, *runtime* (soporte de tiempo de ejecución) en el servidor y *middleware* para la creación y el despliegue de las soluciones Java.

La plataforma Java 2 se divide en tres principales plataformas de desarrollo y *runtime*, cada una enfocada a un tipo de solución específica. La edición estándar de la plataforma Java 2 (J2SE) está diseñada para apoyar la creación de aplicaciones cliente-servidor , mientras que la edición micro (J2ME) se especializa hacia las aplicaciones que funcionan en los

dispositivos móviles. La edición empresarial de la plataforma Java 2 (J2EE) se construye para apoyar las soluciones de mayor tamaño, distribuidas y Web.

La plataforma de desarrollo J2EE consiste una serie de componentes o piezas que pueden ser utilizadas para montar soluciones Web. La figura 7.1 representa las capas subyacentes proporcionadas por la plataforma de J2EE que apoyan una solución orientada al servicio J2EE, abstrayendo la relación entre los componentes de la plataforma de J2EE.

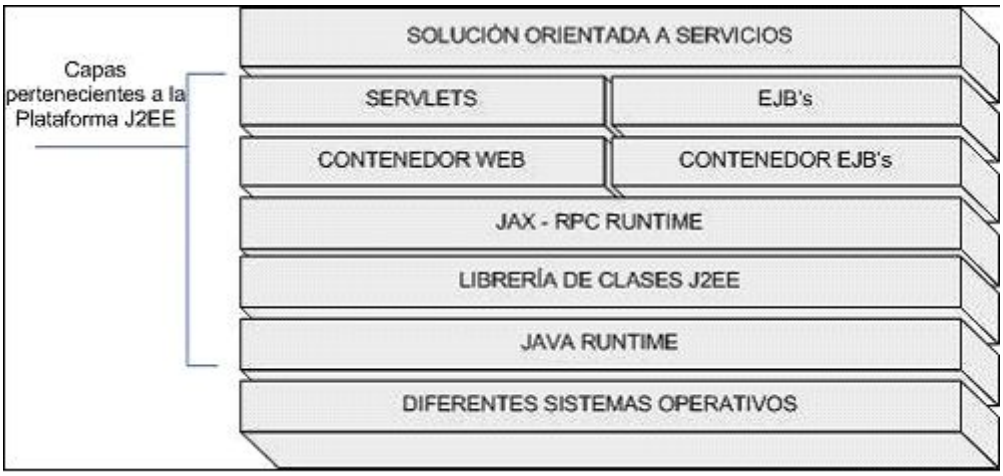

Figura 2-1: Plataforma J2EE para orientación a servicios.

Las capas Servlets, EJBs y Web más el contenedor de capas EJB (así como el runtime de JAX-RPC) se relacionan con las capas de tecnología Web y de los componentes básicos de una arquitectura de SOA (Aplicación de Software, APIS, runtime, Sistema Operativo). La forma de interacción e incorporación de estos componentes y la tecnología Web dependerá de la estrategia de ejecución definida para implementar la arquitectura.

2.3.2 Componentes de la Arquitectura.

Las soluciones J2EE se generan por la relación entre diversos componentes utilizados para construir usos de la Web, entre los cuales se encuentran:

Java Server Pages (JSPs) generado dinámicamente a partir de un Web server. Los JSPs existen como archivos de texto codificados en java, pudiendo contener lenguaje HTML o javascript embebido.

Struts, es una extensión de J2EE que permite el desarrollo de las interfaces de los sitios Web con los usuario (Modelo vista-control), facilitando la navegación.

Java Servlets, estos componentes también residen en el web server y se utilizan para procesar intercambios entre la solicitud y la respuesta del HTTP. Al igual que los JSPs, los servlets son programas compilados.

Enterprice JavaBeans (EJBs), son los componentes de negocio que realizan el grueso del proceso dentro de los ambientes de producción de la empresa. Se despliegan en servidores dedicados y pueden adoptar características de middleware, tales como apoyo a las transacciones.

Mientras que los dos primeros componentes son los más importantes para establecer la capa de presentación de una solución orientada al servicio, los dos últimos se utilizan comúnmente para realizar servicios Web.

2.3.3 Ambientes Runtime.

El ambiente J2EE confía en el *runtime* de Java para procesar los componentes base de cualquier solución de J2EE. En apoyo de servicios Web, J2EE proporciona las capas *runtime* adicionales que, alternadamente, proveen los servicios de APIs específicas. Por su parte, el *runtime* de JAX-RPC establece servicios fundamentales, que incluyen el soporte para establecer la comunicación entre SOA y el proceso de WSDL.

Además, la implementación de J2EE proveen dos tipos de contenedores de componentes que proporcionan un ambiente *host* orientado a los usos de los servicios Web basados generalmente en EJB o *servlet*.

El contenedor EJB está diseñado específicamente para recibir componentes de EJB. Proporcionan una serie de servicios a nivel de negocio dispuestos de manera tal que puedan ser usados por todos los EJBs que participan en la ejecución distribuida de una tarea del

negocio. Algunos ejemplos de estos servicios incluyen la administración de transacciones, de concurrencia y el nivel de seguridad de una operación.

El contenedor Web, puede ser considerado como una extensión de un *Web server* y se utiliza como un Web *host* de Java que contiene aplicaciones en JSP o componentes del *servlet* de Java. Los contenedores Web proporcionan servicios *runtime* orientados hacia el procesamiento de las solicitudes de los JSP y de las instancias del *servlet*.

Los EJB y los contenedores del Web pueden recibir servicios basados en EJB o *servlet* orientados a servicios Web J2EE. La ejecución del servicio Web en ambos contenedores es soportada por servicios *runtime* de JAX-RPC.

J2EE contiene vario APIs para las funciones de programación en apoyo de servicios del Web. Las clases que apoyan este APIs se organizan en una serie de paquetes. Aquí está algo del APIs relevante al edificio SOA. Java API para XML que procesa (JAXP) este API se utiliza para procesar el contenido del documento de XML usando un número de programas de análisis disponibles. El modelo del objeto del documento (DOM) y el API simple para los modelos obedientes de XML (SAX) se apoyan, tan bien como la capacidad de transformar y de validar documentos de XML usando stylesheets de XSLT y esquemas de XSD.

2.3.4 Lenguajes de Programación.

Como su nombre lo indica, la edición empresarial de la plataforma Java 2 está centrada en el lenguaje de programación Java. Diversos proveedores ofrecen productos propietarios de desarrollo que proporcionan ambientes en los cuales el lenguaje estándar sea Java, pudiendo ser utilizado para construir servicios Web.

2.3.5 APIs

J2EE contiene varios APIs (Interfaz de Programación de Aplicaciones) para apoyar las funciones de programación de servicios Web. Las clases que soportan las APIs se organizan en una serie de paquetes. Algunas de las APIs mas relevantes para la construcción de SOA son las siguientes:

Java API para XML que procesa (JAXP). Esta API se utiliza para procesar el contenido de documentos XML usando un *parsers* (analizador sintáctico) disponible. El modelo del objeto del documento (DOM) y la API simple para XML (SAX) son soportados de tan buena manera como la capacidad de transformar y de validar documentos de XML usando hojas de estilo de XSLT y XSD. Los paquetes de ejemplo incluyen:

javax.xml.parsers, es un paquete que contiene las clases para diversos parsers de DOM y de SAX.

org.w3c.dom and org.xml, los paquetes saxThese ponen a disposición de la industria estándares DOM y modelo de documentos SAX.

javax.xml.transform, es un paquete que proporciona las clases que proveen la transformación de funciones XSLT.

Java API para XML basado en RPC (JAX-RPC) Es la mejor y más popular forma utilizada por SOA para procesar API, soportado intercambio entre RPC y solicitudes de intercambio de documentos de respuesta y transmisiones unidireccionales. Algunos ejemplos de paquetes que soportan esta API incluyen:

javax.xml.rpc and javax.xml.rpc.server: contienen una serie de funciones de básicas para el JAX-RPC API.

javax.xml.rpc.handler and javax.xml.rpc.handler.soap, son funciones para el *runtime* de mensajes, proporcionadas por colecciones de clases.

javax.xml.soap and javax.xml.rpc.soap. Funciones API para procesar el contenido y retrasos de mensajes SOAP.

Java API para los registros de XML (JAXR). Corresponde a una API que ofrece una interfaz estándar para los acceder a los registros de negocio y de servicio. Desarrollado originalmente para los directorios XML, JAXR ahora incluye soporte para UDDI.

javax.xml.registry. Una serie de registros que dan el acceso a las funciones que apoyan el JAXR API.

javax.xml.registry.infomodel Clases que representan objetos dentro de un registro.

Java API para la mensajería de XML (JAXM) asincrónico, la mensajería API de documentos SOA se puede utilizar como una forma de enviar y difundir la transmisión de mensajes (pudiendo además, facilitar intercambios síncronos).

SOA utilizando la API para Java (SAAJ) proporciona una API especializada en la administración de mensajes SOA que requieren elementos adjuntos. El SAAJ API es una implementación que contiene dicha especificación.

La arquitectura de Java para XML, API (JAXB). Esta API proporciona medios para generar las clases de Java desde esquemas XSD y fomenta la abstracción a nivel de desarrollo XML.

Servicio de mensajería Java API (JMS). Es una API centrada en protocolos de mensajeria. Es utilizada en soluciones tradicionales de mensajería *middleware* y para proporcionar características confiables de entrega que no se encuentran en la comunicación típica utilizando HTTP.

De las APIs presentadas, las dos mas utilizadas por SOA son JAX-RPC para administrar la mensajería SOA y JAXP para procesar documentos XML. El dos otros paquetes relevantes para construir la lógica para Web J2EE son javax.ejb y javax.servlet, que proporcionan las APIs fundamentales para desarrollar EJBs y de *servlets*.

2.3.6 Proveedores de servicio.

Según lo mencionado previamente, los servicios Web J2EE generalmente se implementan como *servlets* o componentes EJB. En cada caso es conveniente resolver algunos requisitos y configuraciones de instalación y compilación, de acuerdo a los siguientes puntos:

JAX-RPC servicio *endpoint*. Al construir un servicio Web para usar dentro de un contenedor Web, se desarrollo un servicio JAX-RPC que generalmente se implementa como *servlet* por la lógica subyacente del contenedor Web. Los *servlets* son una representación común de los servicios Web dentro de J2EE y el más conveniente para los servicios que no requieren hacer uso de las características del contenedor de EJB.

EJB servicio endpoint. La alternativa es exponer un EJB como un servicio Web por medio de un EJB servicio endpoint. Esta aproximación es apropiada cuando se quiere encapsular la lógica heredada o cuando las características de *runtime* se encuentran disponibles solamente dentro de un contenedor de EJB cuando se requieren. Para construir un EJB servicio endpoint se requiere que el componente subyacente de EJB sea un tipo específico de EJB llamado *Stateless Session Bean*.

Sin importar la plataforma del proveedor, ambos tipos de servicios Web J2EE son dependientes del *runtime* JAX-RPC y de la API asociada.

2.3.7 Cliente de servicios.

La API JAX-RPC también se puede utilizar para desarrollar requerimientos a clientes de servicio. Proporciona la capacidad de crear tres tipos de *proxies* clientes, de la siguiente forma:

- Generar un *stub*. El *stub* generado (o solo el "stub") es la forma más común de cliente de servicio. Es generado automáticamente por el compilador de JAX-RPC (en tiempo de diseño) consumiendo el proveedor de servicio WSDL, y generando un componente Java equivalente a un *proxy*. Específicamente, el compilador crea un interfaz remota de Java para cada tipo de puerto WSDL que expone los métodos que reflejan las operaciones de WSDL. Luego crea un *stub* basado en los puertos y las interrupciones WSDL construidas. El resultado es un componente *proxy* que puede ser llamado como cualquier otro componente de Java. JAX-RPC debe preocuparse de traducir las comunicaciones entre el *proxy* y el componente de la lógica del negocio solicitado por los mensajes SOA que son transmitidos y recibidos por el proveedor de servicio representado por el WSDL.

- El *proxy* dinámico y las llamadas dinámicas de interfaz, dos variaciones de la generación de *stubs* que también son soportadas. El proxy dinámico es conceptualmente similar, salvo que el *stub* actual no se crea hasta que sus métodos son invocados en *runtime*. Por otra parte, la llamada a una interfaz dinámica pasa por alto la necesidad de

un conjunto de *stub* físicos y permite una interacción completamente dinámica entre un componente Java y una definición de WSDL en *runtime*.

Las últimas opciones son mejores para los ambientes en los cuales las interfaces de servicio tienen mayor probabilidad de cambiar o cuando la interacción entre componentes debe ser determinada dinámicamente. Por ejemplo, si al generar un *stub* se produce una interfaz *proxy* estática, puede resultar inútil cuando cambia la definición correspondiente al WSDL. Entonces, la generación dinámica del proxy evita esta situación.

2.3.8 Servicio de agente.

Para implementar plataformas J2EE generalmente se utilizan numerosos agentes de servicio para realizar una variedad de filtros *runtime*, de procesos, y de dirección de tareas. Un ejemplo común es el uso de los agentes de servicio para procesar cabeceras SOAP.

La API JAX-RPC soporta el procesamiento de cabeceras SOAP, permitiendo la creación de agentes especializados de servicio llamados *handlers*, filtros *runtime* que corresponden a extensiones del ambientes de contenedor J2EE. Los *handlers* pueden procesar las cabeceras de los bloques de SOAP para los mensajes enviados por los clientes de servicio J2EE o para los mensajes recibidos por EJB endpoints y de JAX-RPC endpoints.

Los *handlers* múltiples se pueden utilizar para procesar diversas cabeceras de bloques en el mismo mensaje SOAP. En este caso los *handlers* se concatenan en una secuencia predeterminada (llamada una cadena de *handlers*).

2.4 Soporte .NET para SOA.

El *framework* de .NET es la segunda de las dos plataformas más importantes utilizadas para dar soporte a la arquitectura SOA. Como con la sección anterior, primero se verán las partes principales de la plataforma de .NET, luego sus características de diseño y los principios de su orientación a servicios. Para cada punto se exploran solo las características de .NET que proporcionan la ayuda directa o indirecta a SOA.

2.4.1 Descripción de la Plataforma.

El *framework* de .NET es una plataforma propietaria de *runtime* y de desarrollo de soluciones diseñada para los sistemas operativos de Windows y los productos del servidor. La plataforma de .NET se puede utilizar para entregar una variedad de usos, extendiéndose desde sistemas cliente-servidor y móviles hasta soluciones Web y servicios distribuidos.

La principal componente importante para SOA de .NET es el ambiente de ASP.NET, usado para entregar la capa de la tecnología Web dentro de SOA (y para potenciarla, la extensión los *Web Services Enhancements* (WSE)).

Figura 2-2 muestra las capas típicas sobre las cuales están contraídas las soluciones orientadas a servicios de .NET. En este diagrama ASP.NET mas WSE y las capas de las *assemblies* corresponden a las capas de tecnología Web y de los demás componentes básicos.

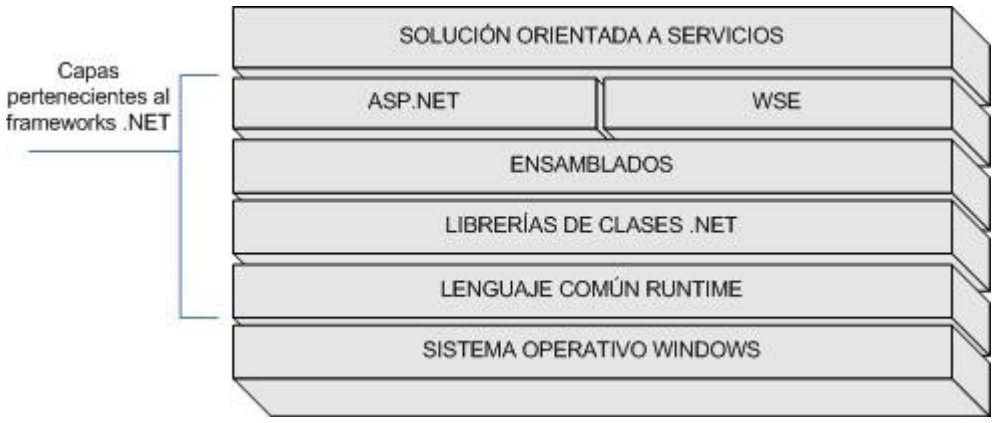

Figura 2-2 Capas relevantes del framework de .NET que se relacionan con SOA.

2.4.2 Componentes de la Arquitectura.

El *framework* de .NET proporciona un ambiente diseñado para entregar diversos tipos de soluciones distribuidas. Los siguientes son los componentes más utilizados en tecnología Web basados en .NET:

Las formas Web de ASP.NET , son las páginas Web construidas dinámicamente que residen en el Web Server y apoyan la creación de formas interactivas en línea con el uso de

una serie de controles en el servidor responsables de generar automáticamente el contenido de la página Web.

Los servicios Web de ASP.NET y las aplicaciones ASP.NET diseñadas como proveedores de servicio también residen en el Web Server.

Los ensamblados son la unidad estándar para procesar la lógica dentro del ambiente.NET. Un ensamblado puede contener múltiples clases que fomentan la reutilización de código usando los principios de orientación a objetos. La lógica del uso de un servicio Web de .NET está generalmente contenida dentro de un ensamblado (pero no necesariamente).

Las formas Web de ASP.NET se pueden utilizar para construir la capa de presentación de una solución orientada al servicio, pero son los últimos dos componentes aquellos de mayor importancia para construir los servicios Web.

2.4.3 Ambientes Runtime.

Los componentes de la arquitectura descritos previamente confían en el *runtime* del lenguaje común (CLR) proporcionado por el framework de .NET. CLR provee una colección de agentes *runtime* que proporcionan un número de servicios para administrar las aplicaciones de .NET, incluyendo soporte de lenguajes, tipos de datos centralizados, ciclo de vida de los objetos y administración de recursos de memoria.

Se pueden agregar varias capas suplementarias *runtime* al CLR. ASP.NET que pos si solo provee de un sistema de servicios *runtime* que establecen comunicación vía HTTP y un ambiente integrado de sistemas de servicio de agentes que incluyen módulos HTTP. Cabe mencionar que el *runtime* establecido de COM proporciona otro sistema de servicios (que incluyen el objeto, las transacciones, los componentes encolados, y activación justo a tiempo) que se ponen a disposición de las aplicaciones .NET.

2.4.4 Lenguajes de Programación.

El framework de .NET proporciona el soporte unificada para una serie de lenguajes de programación, incluyendo Visual Basic, C++, y el C# más reciente. El diseño de las versiones de .NET de estos lenguajes están alineadas con CLR. Esto significa que sin

importar el lenguaje de .NET que se utilice, el código de programación está convertido en un formato estandarizado conocido como *Microsoft Intermediate Language* (MSIL). Es el código de MSIL que eventualmente se ejecuta dentro del CLR.

2.4.5 APIs.

El entorno .NET proporciona el acceso a distintas funciones, que dan un marco de trabajo para determinar las especificaciones de los servicios. Estas funciones están organizadas en librerías contenidas en APIs distribuidas en *Namespaces.* Cada uno de estos *Namespaces* debe ser instanciado explícitamente por la aplicación de manera que sean utilizadas sus características.

A continuación se presentan ejemplos de lo *Namespaces* primarios que proporcionan funcionalidades relevantes para el desarrollo de Servicios Web.

System.Xml Esta colección de clases provee funciones que permiten analizar y procesar documentos en XML. Por ejemplo:

Las Clases: XmlReader y XmlWriter proveen la funcionalidad de leer y generar un archivo XML. Para buscar una parte especifica de un XML las clases XmlNode, XmlElement y XmlAttribute, entregan esta funcionalidad.

System.Web.Services . Esta biblioteca contiene clases que analizan los documentos de la interfaz del Servicio Web así como documentos que pertenecen a la capa de interación del Servidor Web. Por ejemplo:

Los documentos WSDL se encuentran representados en una serie de clases que están en el *Namespace* System.Web.Services.Description. Los protocolos de comunicación (incluyendo los mensajes en formato SOAP) se encuentran en un número de clases bajo el *Namespace* System.Web.Services.Protocols. También se puede mencionar la clase SoapHeader por el System.Web (Namespace de Services.Protocols) que permite el proceso de los estándares de SOAP.

Para complementar la implementación de Servicios Web y soportar el procesamiento de archivos XML, también se pueden mencionar las siguientes clases:

System.Xml.Xsl. Provee funciones para el procesamiento de archivos XML vía transformaciones XSLT

System.Xml.Schema. Este *Namespace* contiene funciones que permiten obtener funciones para archivos XSD (Schema Definition Language).

System.Web.Services.Discovery. Permite obtener los metadatos de los Servicios Web.

Se puede observar que .Net provee de una gran cantidad de funciones que permiten manipular los distintos actores que se encuentran en los APIs de desarrollo.

2.4.6 Proveedores de servicio.

Los proveedores de servicio .NET son los servicios Web que existen como una variación especial de las aplicaciones de ASP.NET, llamados servicios Web de ASP.NET. Se puede reconocer un URL que muestra un servicio del Web de ASP.NET por la extensión de ".asmx" usada para identificar la parte del servicio que actúa como endpoint. Los servicios Web de ASP.NET pueden existir solamente de un archivo de ASMX que contiene código en línea y directorios especiales, pero comúnmente se encuentran comprimidos como un endpoint de ASMX y en un ensamblado compilado que contienen por separado la lógica del negocio.

2.4.7 Cliente de servicios.

Para apoyar la creación de clientes del servicio, .NET proporciona una clase de proxy que reside junto a la lógica del servicio de la aplicación requerida y duplica la interfaz del proveedor de servicio. Esto permite que el solicitante del servicio interactúe con la clase del Proxy en forma local, mientras que delega en forma remota todo el proceso y las actividades que forman el mensaje a la lógica del proxy.

El *proxy* de .NET traduce los métodos de llamadas a requerimientos HTTP y convierte posteriormente los mensajes de respuesta publicados por el proveedor de servicio nuevamente dentro del método de llamadas nativas que retornan requerimientos.

El código detrás de una clase *proxy* se genera automáticamente usando Visual Studio o la línea de comandos de la utilidad WSDL.exe. Cualquier opción deriva en la clase interfaz desde la definición dada por el proveedor de servicio WSDL y después se compila por la clase Proxy una DLL.

Resumen del Capítulo.

Parte de la problemática para plantear un SOA, es definir la estrategia de diseño e implementación de los servicios. Para tomar buenas decisiones, en este capítulo, se menciona cual es la estrategia más adecuada para llevar a la organización una mentalidad orientada al SOA, considerando los procesos de negocio, bajo lo cual se debe definir características básicas como la granularidad del servicio, la reutilización y la interfaz. Los patrones que se mencionan son necesarios dado que son las mejores prácticas para problemáticas comunes.

Las herramientas de desarrollo permiten mejorar los tiempos de implementación, complementadas en varios casos por framework o librerías que tienen implementadas las interfaces y protocolos necesarios para el funcionamiento correcto. En este caso tanto la plataforma .NET y Java permiten la creación rápida de servicios.

3 Servicios Web

En este capítulo está dedicado a explicar los antecedentes, características y funcionamiento de los Servicios Web. Los servicios Web pueden ser utilizados como servicios de manera interna y externa a la organización, permitiendo el acceso a funcionalidades.

3.1 Antecedentes de los servicios Web.

Antes de que aparecieran los servicios Web existían tres técnicas para comunicar aplicaciones:

Se podía escoger una plataforma particular para ofrecer un servicio, como puede ser la plataforma J2SE con la utilización de RMI (*Remote Method Invocation*) [21]para implementar el mecanismo de comunicación.
Se podía utilizar CORBA (*Common Object-Request Broker Arquitectura*)[22].
Se podía utilizar un protocolo definido de comunicación particular entre dos aplicaciones.

El objetivo de una arquitectura SOA es proveer de la transparencia en la localización de servicios Web, es decir, de la posibilidad de utilizar un determinado servicio que se encuentre en cualquier lugar, sin la necesidad de tener que modificar el código existente.

Los servicios Web fueron creados originalmente como un método para compartir recursos en la red. En un entorno donde el aumento constante del número de usuarios demandaba cada vez un mayor número de recursos en la red, surgió la necesidad de facilitar la distribución entre empresas de dichos recursos para satisfacer las necesidades de sus clientes. El resultado fue el desarrollo de una tecnología de sencilla implantación y que era capaz de solucionar los aspectos de disponibilidad e rapidez que se requerían.

La tecnología de los servicios Web permitió estandarizar la comunicación entre distintas plataformas (PC, Mainframe, Mac, etc.) y lenguajes de programación (PHP, C#, Java, etc.). Anteriormente se habían realizado intentos de crear estándares pero fracasaron o no

tuvieron el suficiente éxito, algunos de ellos son DCOM y CORBA, por ser dependientes de la implementación del vendedor DCOM - Microsoft, y CORBA - ORB (a pesar que CORBA de múltiples vendedores pueden operar entre si, hay ciertas limitaciones para aplicaciones de niveles más altos en los cuales se necesite seguridad o administración de transacciones).

Otro gran problema es que se hacía uso de RPC (*Remote Procedure Call*) para realizar la comunicación entre diferentes nodos. Esto, además de presentar ciertos problemas de seguridad, tiene la desventaja de que su implementación en un ambiente como Internet, es casi imposible (muchos firewalls bloquean este tipo de mensajes, lo que hace prácticamente imposible a dos computadoras conectadas por Internet comunicarse). Es por esto que en 1999 se comenzó a plantear un nuevo estándar, el cual terminaría utilizando XML, SOAP, WSDL, y UDDI [24]

Los servicios Web son la estructura de un modelo *On Demand* (bajo demanda). A través de ella todos los recursos de una empresa pueden ser conectados en una sola plataforma integrada y robusta basada en la Web, lo que permite a las compañías reducir la complejidad y aumentar la productividad así como ser más dinámicas y flexibles en la medida que crean, desarrollan y administran sus recursos de tecnología [25].

La iniciativa, que tuvo su origen el año 2002 con miembros de toda la industria de Tecnologías de Información (TI), ha realizado grandes progresos en pro del mejoramiento de la interoperabilidad y publicó especificaciones que permiten a las empresas alinear sus productos de servicios Web. También, en el campo de la seguridad se están identificando las necesidades reales y las áreas en donde se deben realizar los mayores esfuerzos para lograr que los sistemas sean más seguros y estén protegidos contra la intrusión interna y externa.

Durante estos últimos años cada vez más empresas han estado utilizando la tecnología de servicios Web para integrar sus operaciones. Los consultores de TI se encuentran reiteradamente en diversos negocios y empresas con una amplia variedad de plataformas de tecnología incompatibles. Para evitar un gasto significativo en volver a escribir aplicaciones o para adaptarse a servidores o sistemas comunes, es verdaderamente provechoso utilizar

un formato común y un conjunto de reglas comunes para la interacción de sus sistemas. Estas tecnologías deben ser lo suficientemente flexibles para alojar una amplia variedad de plataformas y sistemas operativos. La temprana adopción de servicios Web permite la flexibilidad para interconectar a las empresas utilizando formatos y lenguajes comunes [26].

En la actualidad, esta tecnología ha logrando una aceptación importante excepto para los servicios que implicaban transacciones seguras, debido a que aún se están definiendo los estándares para asegurar el acceso a los servicios Web. Tal es así, que este planteamiento se está empezando a trasladar a la Intranet de las empresas. Así, los servicios Web se están revelando como la tecnología capaz de distribuir los recursos internos entre todos los sistemas, ahorrando costosos desarrollos de integración. Un ejemplo de ello es que muchos sistemas legacy están pasando a ser servicios Web.

3.2 Descripción y funcionamiento de los Servicios Web.

Los servicios Web son componentes de software formados por varios elementos que permiten ser publicados en directorios e invocados para su ejecución por otros programas vía http, generando una respuesta en XML. Está basado en las siguientes tecnologías:

Un formato que describe la interfaz del componente (sus métodos y atributos) basado en XML. Por lo general este formato es el WSDL (*Web Service Description Language*). Este lenguaje está basado en XML, y permite realizar la descripción de los servicios Web definiendo la gramática que se debe usar para permitir su diseño y capacidades (datos, comandos que aceptan o producen), y su publicación en un directorio UDDI (*Universal Description, Discovery and Integration*). En el es posible publicar los servicios Web, permitiendo con ello que los usuarios de ese servicio puedan obtener toda la información necesaria para la invocación y ejecución del servicios Web. Un directorio UDDI ofrece una serie de interfaces que posibilitan tanto la publicación como la obtención de información sobre los servicios Web publicados. La información registrada se clasifica según lo que se desee obtener del servicio:

Información de negocio: acerca de quién publica el servicio.

Información de servicio: descripción del tipo de servicio. Información de enlace: dirección (URL, por ejemplo) para acceder al servicio.

Un protocolo de comunicación basado en mensajes y que permite que una aplicación interaccione (use, instancia, llame, ejecute) al servicio Web. Por lo general este protocolo es SOAP (*Simple Object Access Protocol*). Este protocolo está basado en XML, y sirve para la invocación de los servicios Web a través de un protocolo de transporte, como HTTP. Consta de tres partes: una descripción del contenido del mensaje, unas reglas para la codificación de los tipos de datos en XML y una representación de las llamadas RPC para la invocación y respuestas generadas por el servicio Web.

Un protocolo de transporte que se encargue de transportar los mensajes por Internet. Por lo general este protocolo de transporte es HTTP (*Hiper-Text Transport Protocol*) que es exactamente el mismo utilizado para navegar por la Web. Cabe destacar que los servicios Web no fueron pensados para un protocolo en particular, es decir, que es posible utilizar SOAP sobre algún otro protocolo de Internet (SMTP, FTP, etc.). Se utiliza principalmente HTTP por ser un protocolo ampliamente difundido y que se encuentra menos restringido por firewalls (generalmente se bloquean puertos como el FTP, pero el HTTP es muy probable que no este bloqueado).

3.2.1 Funcionamiento de los Servicios Web.

En el caso que una empresa requiera desarrollar una serie de servicios Web que hagan visibles el resto de aplicaciones corporativas o, con miras a un alcance público desde Internet a una serie de funciones asociadas a las aplicaciones de facturación y cobros de sus clientes, deberá realizar las siguientes actividades [28]

1.- Implementación. En primer lugar se desarrolla la lógica que se quiere ofrecer. Si, por ejemplo, se requiere crear un servicio que retorne el estado (o los estados) de una factura de un cliente, se implementa la lógica generando una consulta a la base de datos entregando los filtros que serán recibidos como parámetros de entrada del servicio creado. Se genera además un fichero WSDL que describe las funcionalidades del servicio, protocolo de transporte y dirección para su invocación.

2.- Publicación. A continuación se publica el servicio en un directorio UDDI, lo que significa hacer público que el servicio ya se encuentra disponible para su acceso. Utilizando la API del directorio para introducir la información de negocio (páginas blancas), la información de servicio (páginas verdes) y la especificación del servicio (páginas amarillas), introduciendo el fichero WSDL de descripción del servicio Web.

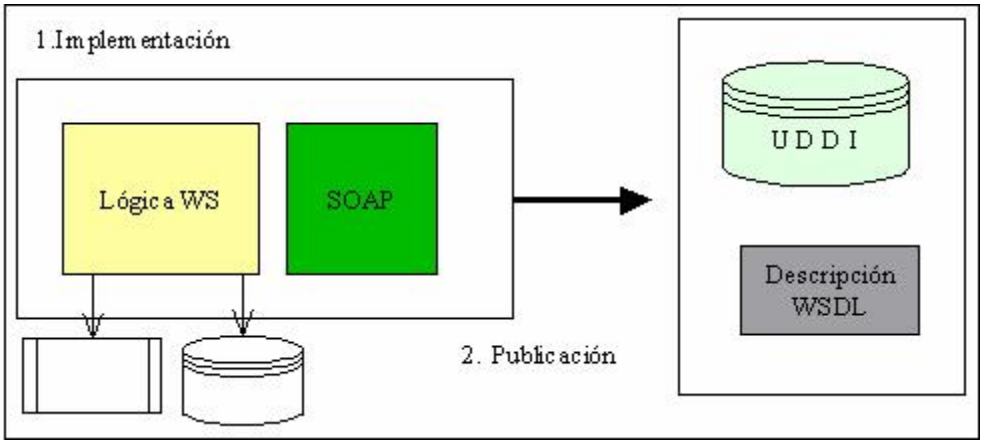

Figura 3-1: Diagrama general de publicación de un Web Services.

Una vez efectuada la publicación, los usuarios pueden utilizar el recurso, en ese caso proceden a:

a) *Obtener la información del servicio*. Se busca el servicio Web realizando una consulta al Directorio UDDI.Para realizar la búsqueda se envía al directorio un mensaje SOAP específico. Como resultado se creará una instancia en el cliente capaz de invocar al servicio Web utilizando la información obtenida en el directorio, que indica cómo acceder, con qué sintaxis y qué protocolo usar para ello.

b). *Ejecutar el servicio.* Se invoca al servicio Web con la información obtenida mediante el encapsulado de mensajes definidos por el protocolo SOAP.

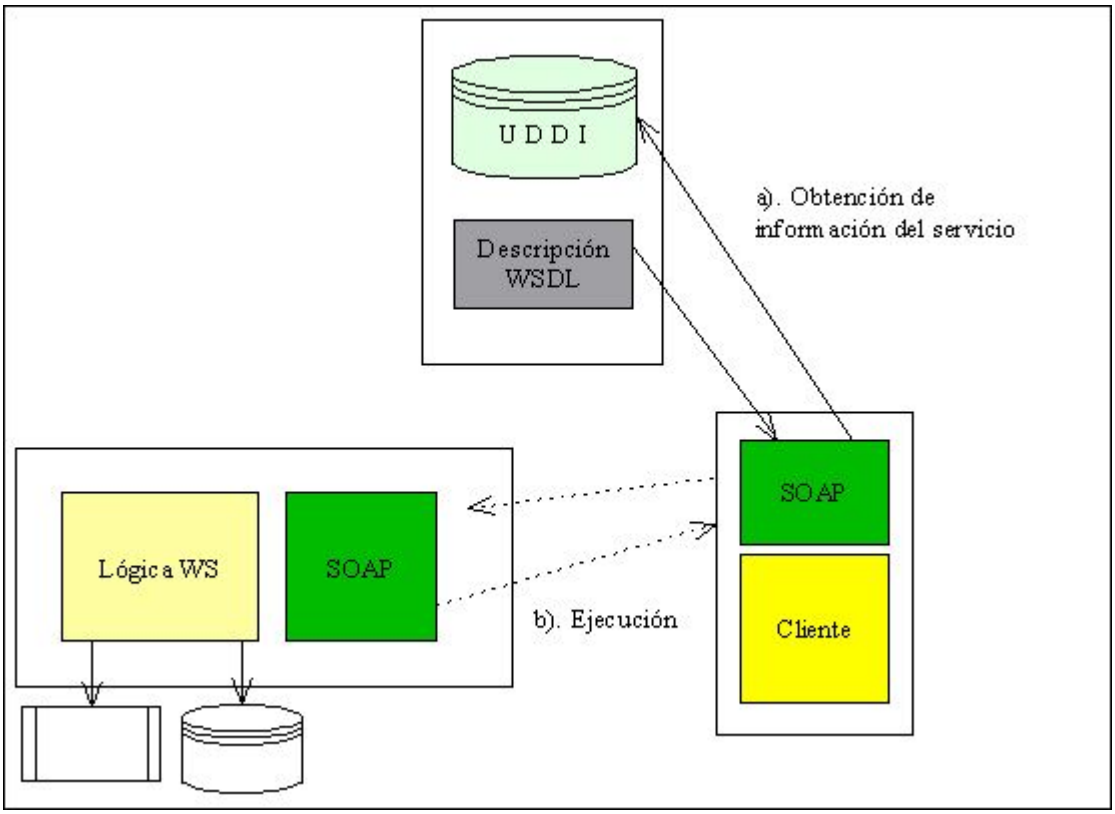

Figura 3-2: Diagrama general de ejecución de un Web Services..

Los servicios Web, no son por tanto aplicaciones con una interfaz gráfica con la que las personas puedan interaccionar, sino que son software accesible en interne t (o en redes privadas que usen tecnologías Internet) por otras aplicaciones. De esta forma es posible desarrollar aplicaciones que hagan uso de otras aplicaciones que estén disponibles en Internet interaccionando con ellas [30]

Un típico ejemplo podría ser un servicios Web al que se le pudiese preguntar por una empresa y que retorne en tiempo real el valor al que están cotizando las acciones de dicha compañía. De esta forma cualquier aplicación (ya sea Web o de escritorio) que quiera mostrar esta información sólo tendría que solicitarla a través de Internet al servicio Web cuando la requiera.

3.2.2 Beneficios de utilizar Servicios Web.

En la actualidad, los desafíos de los negocios definen las necesidades de lo que se denomina una plataforma de próxima generación que puede estar abierta a diversos y

distintos sistemas mientras se adaptan a nuevas plataformas. Una plataforma que haga más eficiente las tareas de integración mediante el máximo aprovechamiento de la potencia de los servicios Web para construir aplicaciones on demand basadas en la red; que acelere las tareas de desarrollo de aplicación on demand, permitiendo la flexibilidad necesaria de estas aplicaciones, en tiempo real.

Por regla general, la implantación de las aplicaciones sirve para resolver una necesidad de negocio o de relaciones entre empleados, proveedores y clientes. Se selecciona la plataforma tecnológica más acorde a las necesidades o requerimientos y se realiza un proyecto de implantación que integre los nuevos sistemas con el resto de procesos de la empresa.

Ahora bien, para realizar dicha integración, se debe disponer de herramientas, que pueden ser:

Soluciones a medida que procesen la generación y recogida de ficheros entre cada sistema, lo que es muy poco escalable ya que cada nueva aplicación requerirá de nuevos y costosos desarrollos.
Aplicación de soluciones *Middleware* (como DCOM, CORBA, etcétera), con altos requisitos para su implantación y costes moderados. Se necesitan, por lo general, plataformas específicas.

El coste de desarrollar software es alto, y la diversidad de las plataformas una realidad desde el inicio de la informática. De hecho conforme más complejas fueron las aplicaciones que las empresas demandaban, más caras era desarrollarlas. Para enfrentarse a esta situación se han seguido diferentes líneas todas ellas encaminadas a reutilizar las aplicaciones ya desarrolladas.

Una de estas propuestas fue la estandarización de lenguajes de programación de tal forma que si cualquiera escribía un programa en C, solo necesitaría un compilador de C en la plataforma específica en la que se quisiera ejecutar la aplicación. Esto en la realidad ha sido difícil de hacer funcionar, porque en seguida surgieron pequeñas diferencias y extensiones de C que hacían difícil transportar una aplicación entre diferentes plataformas. Otra

posibilidad ha sido la que ha desarrollado Sun con Java. Se programa para una plataforma, pero para una plataforma virtual, en este caso para la máquina virtual Java, y en cada plataforma real se implementa una máquina virtual de java, que será la encargada de ejecutar las aplicaciones escritas en Java. Las implementaciones son específicas de cada plataforma pero al ser todas las máquinas virtuales exactamente la misma, los programas escritos en Java deben poder ejecutarse sin ningún problema en todas las plataformas que tengan una máquina virtual de Java. Esta apuesta ha demostrado ser muy útil y funcionar bastante bien.

Sin embargo, ¿qué pasa al tener varias aplicaciones ya desarrolladas en lenguajes propietarios o en plataformas específicas y que deben interactuar entre ellas? El coste de elegir a posteriori un único lenguaje o plataforma y migrarlo al mismo es descabellado en la mayoría de las situaciones, y es aquí donde los servicios Web, así como otras tecnologías, pueden sernos de una grandísima utilidad.

Con los servicios Web se puede reutilizar desarrollos ya utilizados sin importar la plataforma en la que funcionan o el lenguaje en el que están escritos. Los servicios Web se constituyen en una capa adicional a estas aplicaciones de tal forma que pueden interaccionar entre ellas usando para comunicarse tecnologías estándares que han sido desarrolladas en el contexto de Internet.

Resumen del Capítulo.

Los Servicios Web permiten el acceso a funcionalidades desde cualquier ámbito de la organización, desde aplicaciones internas hasta aplicaciones externas.

La implantación de los servicios Web en una empresa no requiere grandes inversiones, ya que aprovecha la infraestructura y las herramientas utilizadas en las aplicaciones. Incluso, llegando aún más lejos, cualquier componente ya desarrollado (EJB, Servlets, objetos COM, etcétera) podrá convertirse en un servicios Web mediante una sencilla adaptación. Además, al estar basados en protocolos ampliamente aceptados, los servicios Web son multiplataforma, facilitando la implantación de los mismos independientemente de la plataforma donde se implante la aplicación.

Aunque muy importante, por los ahorros de costes que permiten, no es ésta la única ventaja que aportan los servicios Web. Su simplicidad, apoyada en unos estándares ampliamente aceptados, permite vislumbrar el valor que esta tecnología podría aportar al negocio de las empresas.

El uso de XML para el formato de los datos hace que cualquier dispositivo que disponga de un parser XML pueda usar los servicios Web. Esto abre las puertas de las aplicaciones a todos los dispositivos que irán haciéndose un hueco en los próximos años: PDAs,Móviles GPRS / UMTS, *Thin Clients*, etcétera.

No es necesario ningún *Middleware* propietario para el intercambio de información entre aplicaciones. En el contexto de la extranet, esto facilita que la comunicación entre los procesos de distintas empresas sea mucho más ágil de implantar, reduciendo con ello el Time to Market de los servicios. Permite, además, la distribución de los procesos en la red. La extensión del uso de los servicios Web, facilita la reutilización de recursos aligerando el peso de las aplicaciones, ya que la lógica de negocio puede repartirse entre distintos sistemas.

4 Caso Práctico

En este capítulo se presenta el caso práctico, donde se aplica lo investigado en los capítulos anteriores, planteando un análisis y diseño para el caso de estudio, definiendo la situación actual, los requerimientos funcionales y no funcionales que debe soportar la arquitectura de software planteada.

Luego se entrega el análisis, basado en casos de uso, y el diseño, para un middelware que está basado en SOA como arquitectura base, para la implantación de nuevas aplicaciones y la integración de aplicaciones actuales.

4.1 Descripción Caso Práctico.

La aplicación de tecnología SOA está orientada hacia una empresa que presta servicios financieros y que tiene sistemas que operan con diversas tecnologías.

Esta empresa está sujeta a los requerimientos propios de un rubro de negocio de naturaleza cambiante y competitiva como son las instituciones financieras, donde la comunicación entre los sistemas que soportan las líneas de negocio y los productos asociados a los distintos segmentos de clientes, es crucial a la hora de generar ofertas de valor y servicios de calidad.

Bajo este escenario, el principal problema de la compañía corresponde a los altos costos en tiempo y dinero generados por la dificultad de integrar aquellos sistemas donde la comunicación no se da de manera natural, debido a la arquitectura que opera en la actualidad. De esta manera, existen sólo algunos sistemas que se comunican entre si, generando la necesidad de extender esta propiedad hacia otras aplicaciones de negocio.

Presentados los antecedentes, el presente capítulo contiene el desarrollo de la solución basada en la adopción de una Arquitectura Orientada a Servicios que sea capaz de comunicar los servicios provistos por los sistemas principales que soportan el negocio de la compañía (sistema PeopleSoft y el sistema propietario). De esta forma, se obtienen

procesos de integración y la generación de un ambiente de comunicación de los sistemas bajo un esquema común, normalizado y que permite escalamiento. Finalmente, la adopción de esta arquitectura permitirá que las distintas aplicaciones se desentiendan de las características específicas de integración.

4.2 Situación Actual Institución Financiera.

En términos de Arquitectura de Sistemas, la Institución Financiera donde se aplicará la tecnología SOA, dispone de sistemas propietarios en tecnología J2EE y Visual Basic con Oracle y ASP, además de un ERP PeopleSoft. Estos sistemas son considerados aplicativos ya están destinados a apoyar el día a día de la empresa. Existen otros sistemas, como por ejemplo el servidor Exchange para manejar cuentas de usuario y correos, que entre otras herramientas, apoyan la gestión administrativa, motivo por el cual no se incluirán dentro de la nueva Arquitectura.

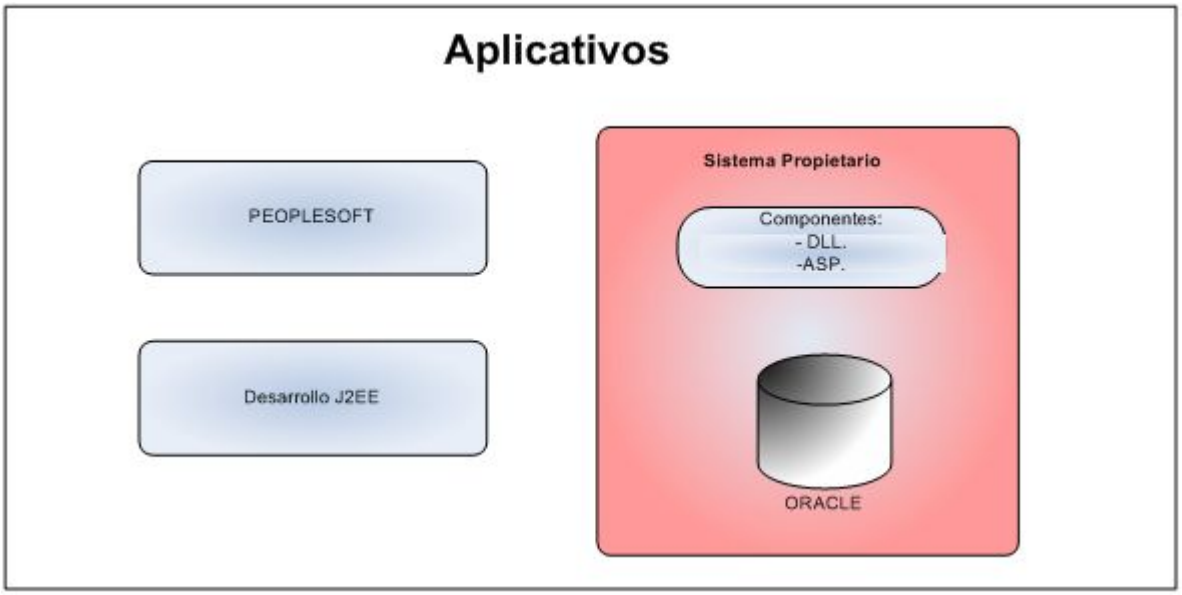

Figura 4-1: Aplicaciones Actuales.

4.2.1 Arquitectura de las aplicaciones actuales.

La, Figura 4-1 muestra los sistemas considerados aplicativos, donde se puede ver el sistema propietario responsable del *core* del negocio. Este sistema se debe integrar con los demás,

dejando disponibles ciertas funciones que eviten generar duplicidad de información relevante.

PeopleSoft

Es un ERP que permite generar servicios y flujos que pueden ser comunicados por una vía estándar con cualquier otro sistema. Esta vía estándar es a base de archivos XML, enviándolo a través de colas JMS.

Sistema Propietario

El sistema propietario tiene como características principales:

Interfaz cliente de páginas Web desarrolladas en ASP.
Lógica de negocio distribuida en base a DLL's desarrolladas en Visual Basic.
Procedimientos Almacenados implementados en PL/SQL para Oracle.

El desarrollo J2EE existente está desarrollado en Java sobre un servidor de Aplicaciones WebSphere 5.1. con servidor de base de datos Oracle o SQL Server.

Áreas o procesos impactados

En referencia a como afecta la implantación de una Arquitectura orientada a Servicios a los proyectos tecnológicos de la organización, existen dos puntos de vista; el primero corresponde al impacto generado en proyectos ya terminados o en ejecución, y el segundo a los proyectos nuevos.

En el primer escenario la solución no impactará a esos proyectos, porque no está dentro de los objetivos de esta etapa de diseño e implantación de SOA comunicar los sistemas actuales entre sí, no obstante se debe considerar la integración a partir de los componentes que queden disponibles a partir de este proyecto.

En segundo caso existe un fuerte impacto dado por la necesidad de cambiar la estrategia de ejecución e implementación de la arquitectura, definiendo servicios que correspondan a las

reglas de negocio encapsulados en componentes que permitan la reutilización y escalabilidad para futuros proyectos.

4.3 Requerimientos Caso Práctico.

Dada la situación actual y en conjunto con la investigación realizada, el caso práctico se enfoca a los siguientes requerimientos para montar la nueva Arquitectura Orientada a Servicios. Estos requerimientos están divididos en dos categorías, Requerimientos Funcionales y Requerimientos No Funcionales, en la primera se entrega un listado que determina las funcionalidades que debe tener la nueva Arquitectura. Para la segunda categoría se plantea los requerimientos que no son funcionalidades del sistema y que son relevantes para el mismo.

4.3.1 Requerimientos Funcionales.

A continuación se presenta los Requerimientos Funcionales, con los que la nueva arquitectura debe cumplir al finalizar el proyecto.

Código	**Nombre**	**Descripción**
RF-1.1	Administrar Servicios.	Permite ingresar, modificar y eliminar los servicios que se encuentran disponibles en el catálogo de servicios.
RF-1.2	Ejecutar Servicios.	Se refiere a la ejecución de Servicios, ya sea de prueba al crear uno nuevo o en tiempo de ejecución.
RF-1.3	Generar un Bus de Comunicación.	A través de mensajería se permite la comunicación con la nueva arquitectura, donde se recibe un mensaje de petición y la respuesta va en un mensaje.
RF-1.4	Monitoreo	Permite monitorear la disponibilidad de

		mensajes, bases de datos o servicios.
RF-1.5	Generar Reportes	A partir de la funcionalidad de monitoreo se plantea la necesidad de generar un módulo de reportes que permita visualizar la disponibilidad y el nivel de fallas que ocurren en la arquitectura.
RF-1.6	Administrar Usuarios	Esta funcionalidad permite la administrar los usuarios que van a interactuar con los servicios y con la aplicación de administración.
RF-1.7	Consolidar Datos de Clientes de Póliza	Esta funcionalidad permite generar un servicio que consolide en los sistemas propietarios la información relativa a un cliente de una póliza.
RF-1.8	Administrar Gestión de Mandatos	Servicio que realiza la actualización de los mandatos en los sistemas propietarios.

Tabla 4-1: Descripción Requerimientos Funcionales.

4.3.2 Requerimientos No funcionales.

El objetivo de plantear los requerimientos no funcionales corresponde a condiciones que el sistema debe cumplir para dar valor y satisfacer las necesidades de los actores. Es de esta forma que los requerimientos no funcionales corresponden a características que debe tener el sistema para cumplir de manera eficiente y eficaz los requerimientos funcionales. A continuación se detallan los requerimientos no funcionales que debe cumplir la arquitectura:

Requerimientos de seguridad.

El sistema debe tener perfilamiento a nivel de administración de los servicios. De esta manera se permite que sólo usuarios autorizados puedan interactuar con los servicios. A

nivel de ejecución de servicios estos también debe poseer perfilamiento, donde se permitirá la realización de la llamada a través de usuarios plenamente identificados.

Requerimientos de Portabilidad.

La consola de administración de los servicios, debe ser diseñada de manera que se pueda tener acceso desde cualquier computador que posea los mínimos requerimientos.

Por otro lado la llamada y respuesta entregada por los servicios debe ser implementada en un lenguaje estándar (XML), de manera que los servicios sean independientes a como el aplicativo los llaman.

Requerimientos de Mantención.

La aplicación es paramétrica, lo que permite al usuario con los privilegios necesarios, modificar los parámetros que están siendo utilizados sin necesidad de intervenir el código.

Requerimientos operacionales.

La aplicación debe considerar un interfaz de usuario amistosa y fácil de usar, debe ser intuitiva y de rápido acceso a la funcionalidad deseada. Además debe poseer óptimos tiempos de respuesta ante los requerimientos de los usuarios.

La actualización de los datos modificados por una operación se debe ver reflejada inmediatamente en la aplicación. El sistema es capaz de mitigar inconsistencia de datos producto de operaciones que involucren datos críticos para el normal funcionamiento del sistema. Se define que para establecer una interacción válida con el Sistema Propietario se deben crear aplicaciones Java (EJB’s) capaces de invocar un Procedimiento Almacenado, con conexión vía JDBC, para interactuar con este sistema. Estos servicios están disponibles a través del catalogo de servicios quedando utilizables para su ejecución por cualquier aplicación.

El sistema debe ser creado bajo un servidor de aplicaciones WebSpehere 5.1. La arquitectura es interna a la organización, en esta etapa no se considera comunicación entre otras compañías, aunque dentro del diseño se debe considerar este punto.

Requerimientos escalabilidad.

El sistema se puede ampliar añadiendo más funcionalidad y/o ofreciendo la funcionalidad actual a más usuarios. Se considera que la aplicación es modular y que permite crecer en forma ordenada y de acuerdo a los requerimientos

Requerimientos de acoplamiento.

Los servicios que entrega la aplicación son dependientes entre sí, es decir, los cambios en una función afectan al resto de servicios que utilizan los operadores, esto es a nivel del aplicativo para administrar los servicios. Por definición los servicios tienen bajo acoplamiento y la interacción entre ellos es independiente de la arquitectura.

4.4 Análisis Caso Práctico.

Considerando los requerimientos funcionales la construcción de la nueva arquitectura se puede separar en dos grandes capas. La primera referente a la aplicación que debe soportar el ingreso y modificación de los servicios. Por otro lado, se tiene el módulo que soporta todo lo que es la llamada al servicio, su ejecución y registros de eventos. Con motivo de visualizar las funcionalidades relacionadas con los requerimientos funcionales planteados en el punto 4.3.1, se realiza un análisis a nivel de casos de uso, visualizando tanto diagramas como el respectivo formato narrativo.

4.4.1 Caso de Uso: “Contexto”.

Este caso de uso busca reflejar el contexto general de la nueva arquitectura que cubra con los requerimientos. En la Figura 4-2 se puede apreciar el Diagrama de Caso de Uso. Cabe mencionar que existen dos actores que son los que utilizarían las prestaciones de la arquitectura:

Administrador. Aplicativo.

Donde el rol del “Administrador”, radica en administrar la configuración de la arquitectura, donde destaca la agregación y parametrización de servicios en conjunto con el manejo de usuarios.

Por otro lado el rol “Aplicativo”, se refiere a la aplicación que finalmente requiere de un servicio y lo llama a través de la arquitectura.

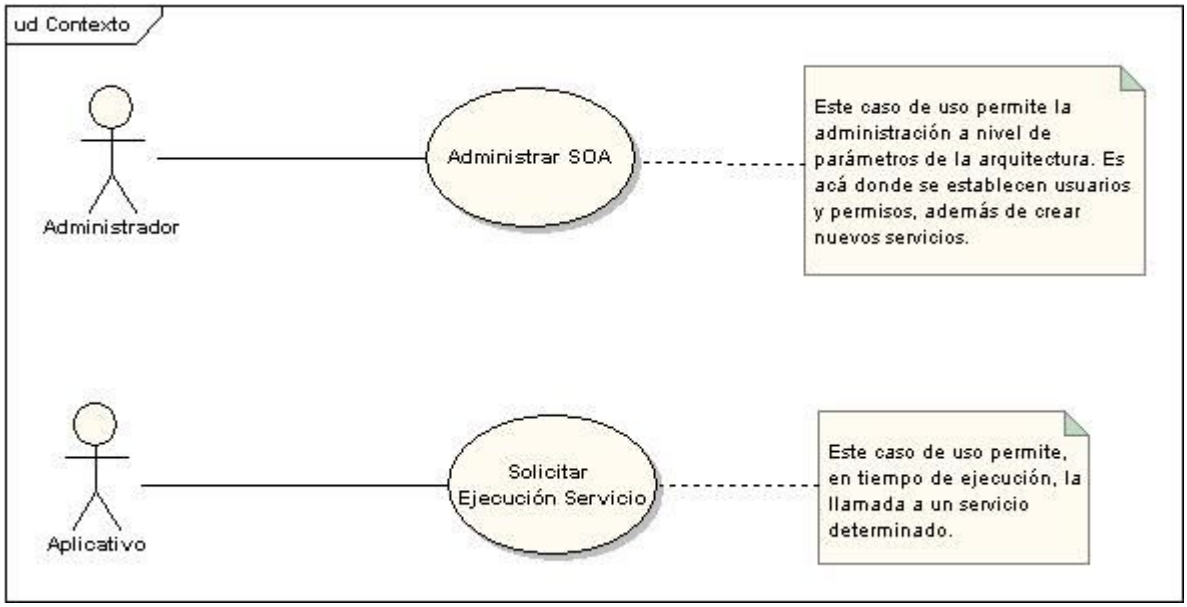

Figura 4-2: Diagrama de Caso de Uso: Contexto.

A continuación se presenta el formato narrativo de los casos de uso, presentados en la, en Figura 4-2 un formato resumido.

Caso de uso	**Administrar SOA**.
Descripción	Este caso de uso es genérico, permite la administración de los servicios, usuarios y configuración de la arquitectura.
Actores	Administrador (Iniciador).
Requerimientos	RF-1.1, RF-1.2, RF-1.4, RF-1.5, RF-1.6.

Precondiciones	
Poscondiciones	
Inclusiones	
Extensiones	Administrar Usuario, Monitorear Arquitectura, Administrar Servicios.

Tabla 4-2: Formato Narrativo Caso de Uso: Administrar SOA.

Caso de uso	**Solicitar Ejecución de Servicio**.
Descripción	Este caso de uso permite la llamada de un servicio solicitada por una aplicación.
Actores	Aplicativo (Iniciador).
Requerimientos	RF-1.3.
Precondiciones	
Poscondiciones	
Inclusiones	Recibir Peticiones.
Extensiones	

Tabla 4-3: Formato Narrativo Caso de Uso: Solicitar Ejecución de Servicios.

4.4.2 Caso de Uso: "Administrar SOA".

Este caso de uso cumple con la funcionalidad de configurar parámetros de la arquitectura. En este caso el rol principal viene dado por el "Administrador", el cual puede agregar usuarios en el sistema, definir sus roles y permisos asociados. Por otro lado es el

administrador el que se preocupa de configurar los aplicativos correspondientes al monitoreo y generar los respectivos reportes.

En la Figura 4-3 se puede apreciar el Diagrama de Caso de Uso, en este caso el administrador (iniciador de este caso de uso) tiene la opción de Administrar Servicios, donde es acá donde se puede agregar, modificar, activar/desactivar o ejecutar un servicio. Al agregar un servicio se debe considerar que se deben incluir todos los parámetros de necesarios para su ejecución. Adicionalmente se considera una funcionalidad que permita probar el servicio, ejecutándolo, de manera de validar la correcta parametrización del servicio instalado.

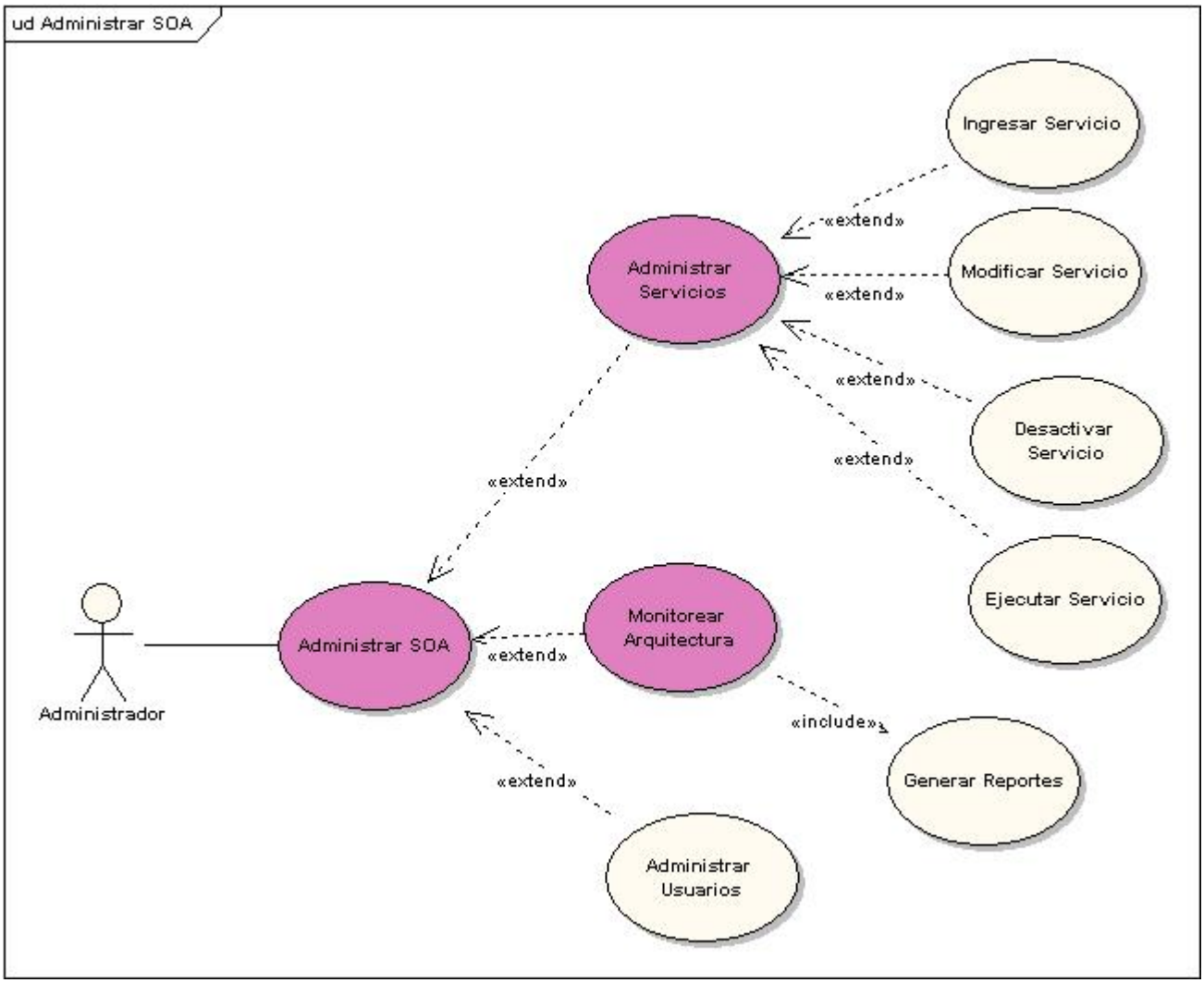

Figura 4-3: Diagrama de Caso de Uso: "Administrar SOA".

4.4.3 Caso de Uso: “Solicitar Ejecución de Servicio”.

Este caso de uso refleja al core de la arquitectura, donde se puede apreciar el funcionamiento de un flujo de llamada para la ejecución de un servicio por parte de un Aplicativo. En este caso el Aplicativo es el actor principal, el cual mediante un mensaje envía una solicitud de ejecución de un servicio. Este mensaje debe considerar, como medio de seguridad, un usuario y un password válidos para la ejecución.

Luego de consultar y validar el servicio y sus parámetros incluidos en el mensaje, se procede a ejecutar el servicio registrando el evento en un log correspondiente al monitoreo de la arquitectura. A partir de de este evento se retorna un mensaje al aplicativo, el cual recibe la información correspondiente a la ejecución del servicio, en el sentido si fue exitoso o falló en algo.

En la, Figura 4-4 se puede apreciar el Diagrama de Casos de Usos, correspondiente al flujo descrito anteriormente, donde una Aplicación llama a un servicio determinado para cumplir con sus procesos de negocios. A continuación de la Figura 4-4 se presenta en formato narrativo resumido, todos los casos de usos incluidos en el diagrama.

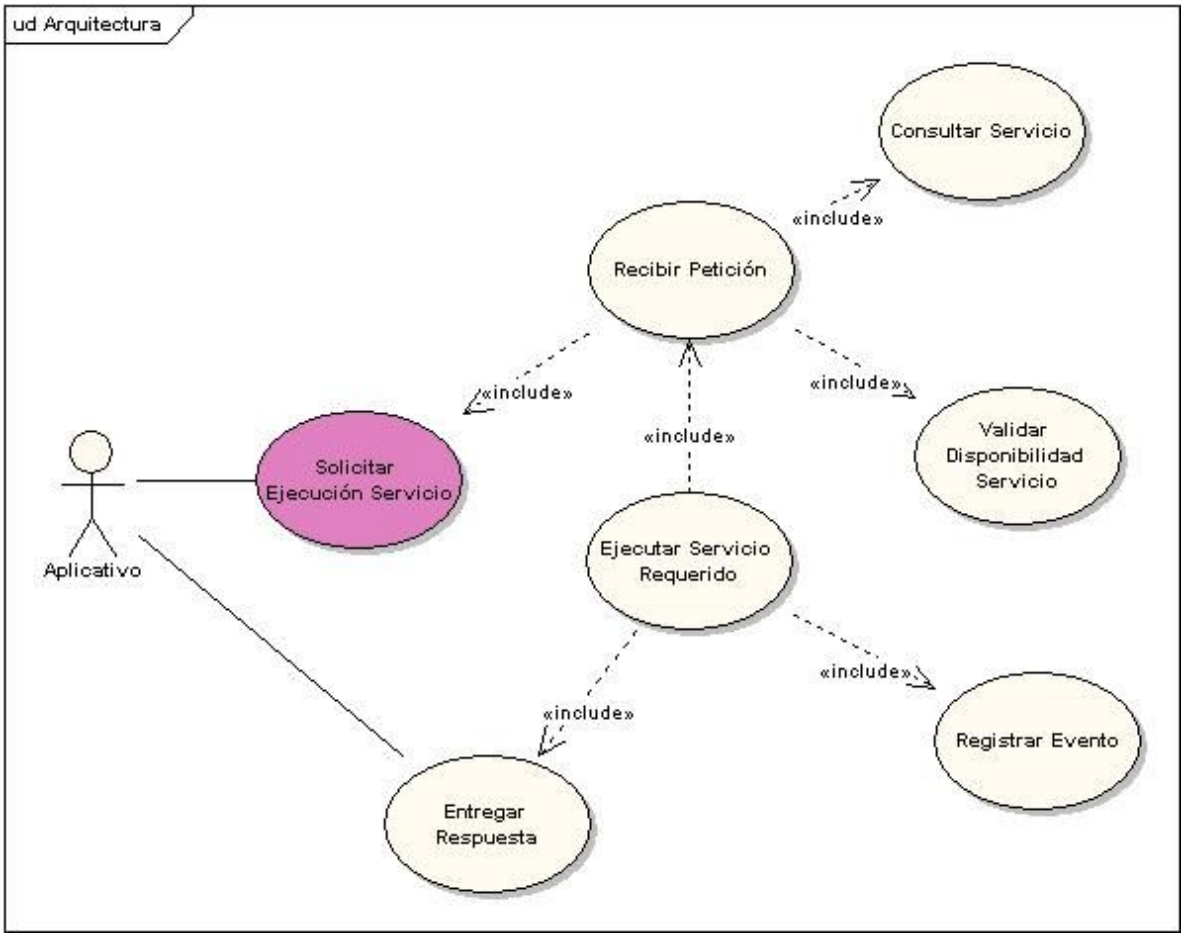

Figura 4-4: Diagrama de Caso de Uso: "Solicitar Ejecución Servicio".

Caso de uso	**Recibir Petición**.
Descripción	Este caso de uso recibe una petición para ejecutar un servicio, esta petición es en forma de mensaje donde viene el servicio a ejecutar, existe un proceso que debe consultar cual es el servicio y como se debe ejecutar.
Actores	Aplicativo (Iniciador).
Requerimientos	RF-1.3.
Precondiciones	El servicio debe estar creado previamente en el sistema.

Poscondiciones	
Inclusiones	Consultar Servicio, Validar Disponibilidad de Servicio
Extensiones	

Tabla 4-4: Formato Narrativo Caso de Uso: Recibir Petición.

Caso de uso	**Consultar Servicio**.
Descripción	Este caso de uso permite consultar sobre la formad de ejecutar un servicio extrayendo parámetros de entrada, parámetros de salida y llamada para su ejecución.
Actores	Aplicativo.
Requerimientos	RF-1.3.
Precondiciones	Debe haber llegado una petición de ejecución de servicios.
Poscondiciones	
Inclusiones	
Extensiones	

Tabla 4-5: Formato Narrativo Caso de Uso: Consultar Servicio.

Caso de uso	**Validar Disponibilidad del Servicio**.
Descripción	Este caso de uso permite validar el estado de un servicio, si está activado o si sus componentes están disponibles.
Actores	Aplicativo.

Requerimientos	RF-1.3.
Precondiciones	El servicio debe de haber sido consultado previamente.
Poscondiciones	
Inclusiones	
Extensiones	

Tabla 4-6: Formato Narrativo Caso de Uso: Validar Disponibilidad del Servicio.

Caso de uso	**Ejecutar Servicio Requerido**.
Descripción	Este caso de uso ejecuta el servicio, tomando la forma de llamarlo con los parámetros de entrada que vienen en el mensaje de petición.
Actores	Aplicativo.
Requerimientos	RF-1.3.
Precondiciones	El servicio debe haber sido validado anteriormente.
Poscondiciones	Petición atendida con éxito o con error
Inclusiones	Recibir Petición, Registrar Evento.
Extensiones	

Tabla 4-7: Formato Narrativo Caso de Uso: Ejecutar Servicio Requerido.

Caso de uso	**Registrar Evento**.
Descripción	Este caso de uso registra en base de datos el estado de la solicitud de servicio, ya sea exitoso o con errores.

Actores	Aplicativo.
Requerimientos	RF-1.3.
Precondiciones	El servicio debe haber sido ejecutado anteriormente.
Poscondiciones	
Inclusiones	
Extensiones	

Tabla 4-8:Formato Narrativo Caso de Uso: Registrar Eventos.

Caso de uso	**Entregar Respuesta**.
Descripción	Entrega la respuesta de la llamada al servicio al aplicativo que realizó la petición de ejecución del servicio. Esta respuesta puede ser un estado o un conjunto de datos dependiendo del tipo de servicio.
Actores	Aplicativo.
Requerimientos	RF-1.3.
Precondiciones	El servicio debe haber sido ejecutado anteriormente.
Poscondiciones	Solicitud atendida por la arquitectura.
Inclusiones	Ejecutar Servicio Requerido.
Extensiones	

Tabla 4-9: Formato Narrativo Caso de Uso: Entregar Respuesta.

4.5 Diseño Caso Práctico.

Para enfrentar el diseño requerido para implementar el caso práctico definido anteriormente, se considera los siguientes diagramas:

1. Diagrama de Secuencia.

2. Diagrama de Clases.

3. Modelo de Datos.

Con estos artefactos se espera cubrir el diseño de la arquitectura, de esta forma se espera concluir con las especificaciones necesarias, que cubran el espectro de requerimientos funcionales y no funcionales, para su implementación.

4.5.1 Diagramas de Secuencia.

Los diagramas de secuencia son construidos a partir de lo casos de uso definidos en la etapa de análisis.

4.5.1.1 Solicitar Ejecución de Servicio.

En este caso se visualiza el comportamiento de la arquitectura en el momento de ser solicitado, por un aplicativo, la ejecución de un servicio.

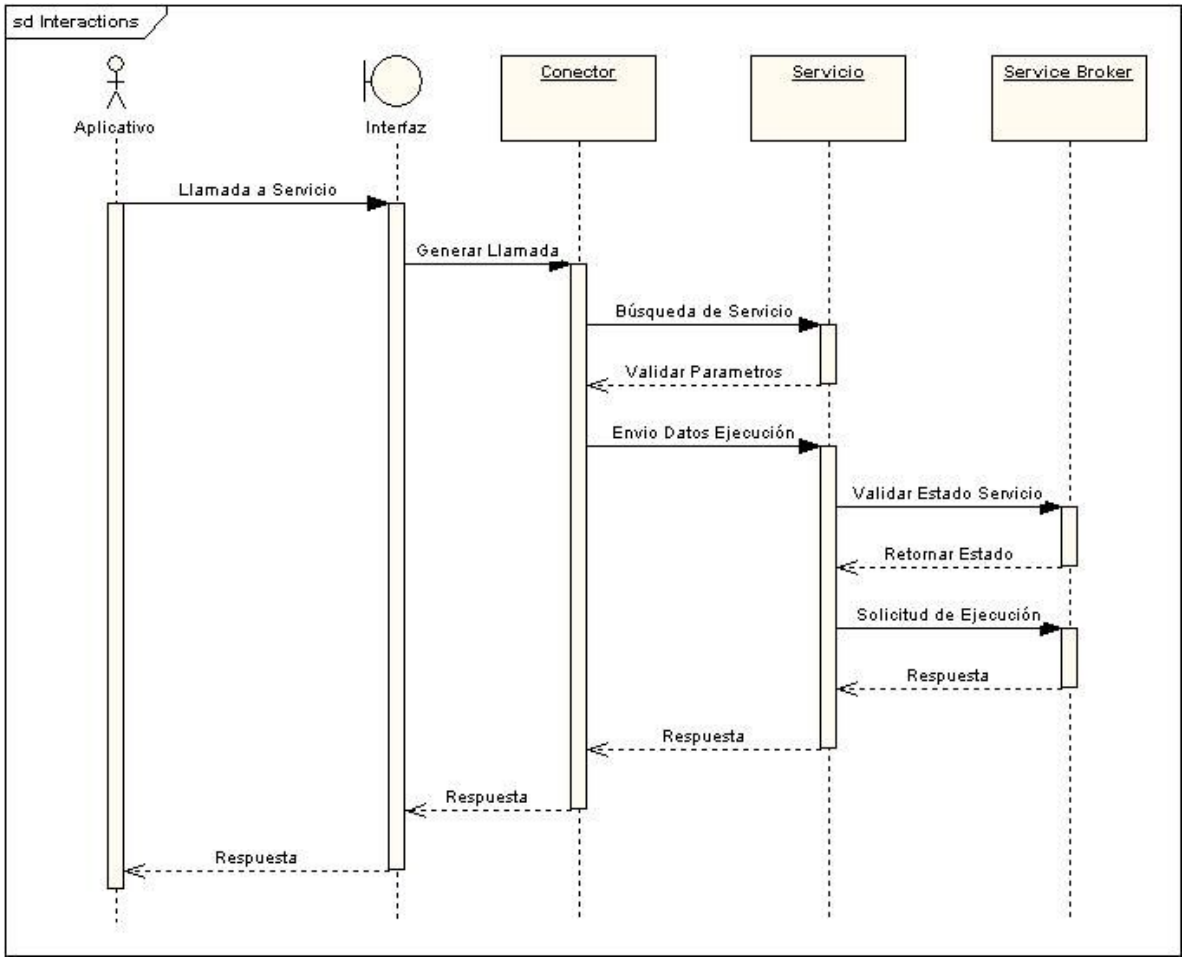

Figura 4-5: Diagrama de Secuencia: Solicitar Ejecución de Servicio.

4.5.2 Modelo de Clases.

Para soportar las funcionalidades planteadas para la nueva arquitectura se presenta un diagrama de clases, en un nivel de contexto el cual refleja las clases que se deben considerar para el funcionamiento de la arquitectura. Cabe considerar que el nodo a implementar es el “Service Broker” ya que es en ese nivel donde funciona la arquitectura.

En la Figura 4-6, se puede apreciar el Diagrama de Clases “Nivel 0”, con este diagrama se plantea el funcionamiento a nivel general de la nueva arquitectura. Existe un nodo denominado “Service Consumer”, en el que funcionan todos los aplicativos que requieran de la ejecución de un servicio determinado, bajo este esquema el Sistema Consumidor envía una petición en forma de mensaje JMS, existe un listener escuchando ese mensaje

ante lo cual lo envía al “Bus Arquitectura”, donde ya se sabe que tipo de servicio se debe ejecutar.

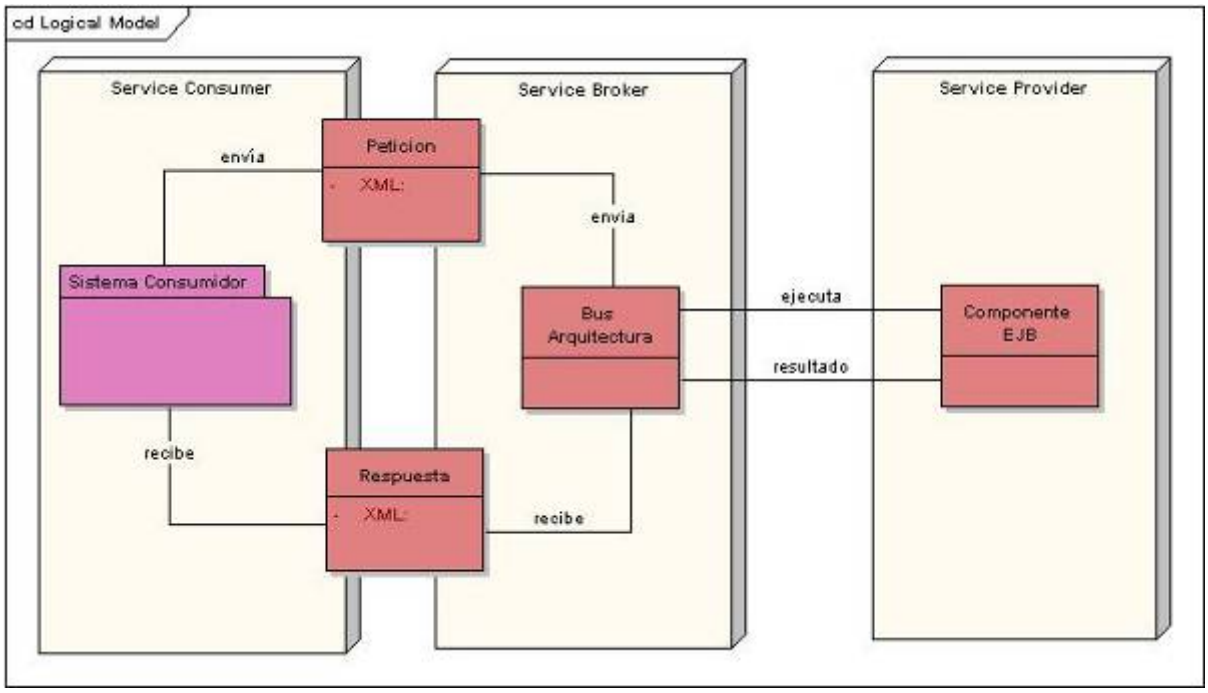

Figura 4-6: Diagrama de Clases Nivel 0.

Esta clase ejecuta el servicio, el cual está encapsulado en un componente EJB, de manera de controlar las transacciones a través del mismo servidor. Este componente, donde ya es propiamente tal el servicio, se encuentra en el nodo “Service Provider” y retorna el valor de la operación en forma de mensaje JMS hacia el “Bus Arquitectura”, el cual lo devuelve a la capa del Service Consumer, retornando el valor del mensaje procesado.

En un nivel de detalle más profundo se muestra el diagrama de Clases Nivel 1, el cual explica con más detalle, introduciendo en los componentes que al final serán implementados dentro de la arquitectura. Este diagrama se puede visualizar en la Figura 4 7, el objetivo es visualizar como se comporta un requerimiento a través de las clases que componen el sistema. Además de se puede apreciar de manera preliminar las clases y paquetes, con los componentes que posee la arquitectura.

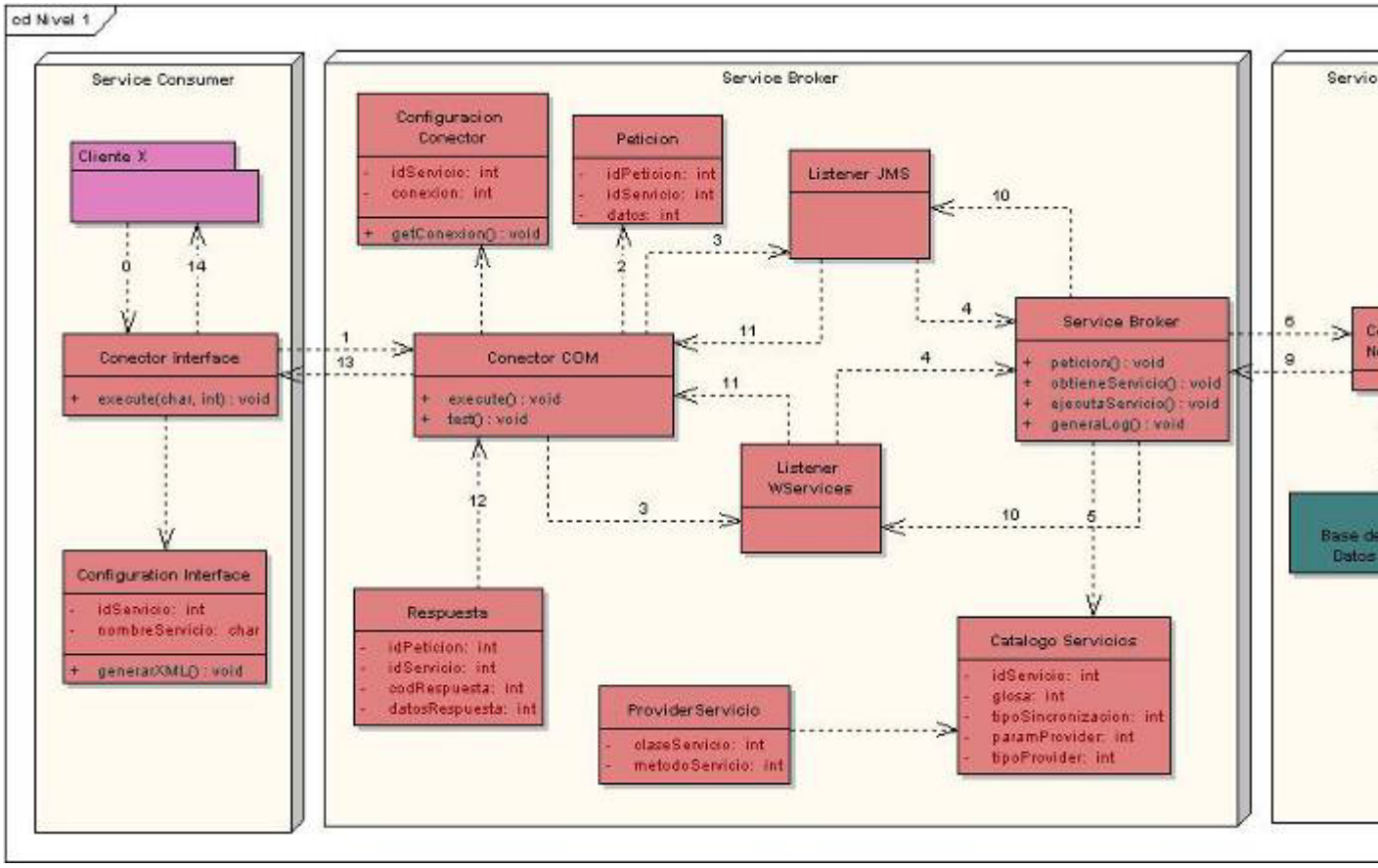

Figura 4-7: Diagrama de Clases Nivel 1.

4.5.2.1 Módulos y Componentes de la Arquitectura.

Los módulos propios del Sistema Solución son:

1. Conector.
2. Service Broker.
3. Catalogo de Servicios.
4. Listener JMS.
5. Listener Webservice.
6. Petición.
7. Respuesta.

8. BaseNegocio.

9. Configuración Conector

4.5.2.1.1 Conector.

API que permite ejecutar un método para llamar al servicio (método “execute()”), este conector puede ser un componente COM para las aplicaciones win32, o una clase Java para las aplicaciones J2EE. El conector es el que establece la comunicación con el Sistema Solución. El método execute() recibe la identificación del servicio (IdServicio), y los parámetros de entrada del servicio en formato XML, devuelve la respuesta en XML. Los parámetros de conexión al sistema deben estar en un archivo de configuración, estos parámetros definen el tipo de listener a utilizar (ListenerJMS, ListenerWService), y los parámetros necesarios para conectarse a cada listener. Además se provee un método de test() que permite probar el ambiente del Sistema Solución.

Las funcionalidades que debe considerar son:

1. Recibir mensaje XML de Entrada

2. Generar ID de petición

3. Configurar forma de conexión, TimeOut, N° max de intentos)

4. Armar mensaje JMS.

5. Enviar mensaje a cola de Entrada Sistema Solución

6. Recibir OK si el mensaje fue entregado exitosamente en Cola Sistema Solución. Sino enviar mensaje a Cola Alarma

7. Recibir mensaje JMS desde Cola de Respuesta

8. Enviar respuesta XML a Conector Interface

4.5.2.1.2 Service Broker.

Sistema que recibe el requerimiento, y según el Catalogo de Servicios delega al *Provider* adecuado, en este caso ejecuta un EJB.

4.5.2.1.3 Catálogo de Servicios.

Archivo XML que define los distintos servicios, tiene 5 campos:

- **idServicio**: identificación del servicio, que viene en mensaje de entrada.

- tipoSincroniza: identifica el tipo de respuesta si es sincronía o asíncrona, si es asíncrona el Sistema Solución devuelve de inmediato una respuesta al conector de que recibió el mensaje, si es sincrona se espera hasta la respuesta del "provider", en este caso se espera la respuesta del ComponenteNegocio.

- **glosa**: descripción del servicio para poder identificarlo rápidamente en el catalogo.

- **tipoProvider**: identifica el tipo tecnología en que se implemento el servicio (EJB, Webservice, o JMS MQSeries).

- **paramProvider**: parámetros necesarios para ejecutar o redireccionar el servicio, para EJB define parámetros ejecución del método (jndi, clase, método), para Webservice corresponde a parámetros de conexión al Webservice (url, serviceName, portName, operationName), y para MQSeries los parámetros de conexión a la Cola (queue, replyQueue).

4.5.2.1.4 Listener JMS.

MDB (*Message Driven Bean*) canal de entrada para peticiones de servicio puede haber mas de uno de forma de no tener un cuello de botella, cada MDB tiene asociada una Cola de Entrada y una de Salida (*Websphere MQ*). Como son parte del sistema pero es un componente que puede cambiar debería estar un grupo (EAR) distinto al núcleo del sistema

de forma de no tener que recompilar y modificar todo el Sistema Solución cada vez que se agregue un nuevo *Listener*, pero estos MDB (archivo EAR) debiera estar en el mismo servidor de aplicaciones del Sistema Solución.

4.5.2.1.5 Listener Webservice.

Webservice como canal de entrada para peticiones de servicio, debería ser uno solo, pero se contempla la posibilidad de tener mas listener de este tipo, por lo tanto también debería estar en un modulo EAR independiente del núcleo del Sistema Solución pero en el mismos servidor de aplicaciones.

4.5.2.1.6 Petición.

Mensaje XML que corresponde a una petición de servicio tiene los siguientes campos:

idServicio: identificación única del servicio, puede ser un correlativo, o un nemotécnico único, ejemplo: “actualizarDireccion”, este debe ser definido para cada nuevo servicio a implementar.

idPeticion: identificador único del mensaje, debiera ser un correlativo único, este va a ser generado automáticamente por el conector, se recomienda utilizar una combinación de *timestamp* (representación en entero del milisegundo actual) mas un correlativo único de 4 dígitos (o un random), este identificador no debe superar el máximo de dígitos de un Long.

datos: los datos de entrada necesarios para realizar el servicio, y que son propios de cada servicio a realizar, debe ser en formato XML, y debe ser definido para cada servicio en cuestión.

4.5.2.1.7 Respuesta.

Mensaje XML que es la respuesta del servicio requerido, puede ser tan simple como un “OK”, es decir que se realizó el servicio, su forma debe ser la siguiente:

idServicio: identificación única del servicio, es el mismo con que se hizo la llamada (petición).

idPeticion: identificador único del mensaje, es el mismo con que se hizo la llamada.

codRespuesta: corresponde a un código numérico que indica por ejemplo si la respuesta es un error (codRespuesta <>0), o indica que el servicio fue procesado exitosamente (codRespuesta=0), los códigos 1-999 están reservados para errores o tipos de respuesta del Sistema Solución, y los código 1000- son propios de cada sistema que atiende (Service Providers).

datosRespuesta: los datos que son el resultado de realizar el servicio, por ejemplo si es un servicio de consulta en este campo debieran venir la información consultada, si el resultado es mas de un campo debe ser entonces en formato XML. Si el resultado es un error (codRespuesta<>0) tanto del sistema, o como del Provider, en este campo vendría un String con la descripción del error.

4.5.2.1.8 Base de Negocio.

Es un EJB entregado por el Sistema Solución del cual deben heredar los componentes de negocio que van a ser "*providers*" (ComponenteNegocio), esto cumple 2 funciones: primero permite que los componentes hereden características generales de los servicios tipo Sistema Solución, y que exista un servicio de prueba básico para cada componente (método test()), este método de prueba recibe el XML de entrada y lo devuelve como salida, esto sumado a que el conector tiene un método test() que llama a este servicio, podemos fácilmente hacer pruebas del ambiente de integración, y nos sirve como servicio "*dummy*" para probar la integración a nivel de clientes sin necesidad de tener desarrollado aun los *providers*.

4.5.2.1.9 Configuración Conector.

Archivo de configuración XML, que permite que para cada servicio (idServicio) se defina la forma de conectarse al Sistema Solución.

4.5.3 Modelo de Base de Datos.

Para soportar la funcionalidad de seguridad y agregación de servicios, se plantea el siguiente esquema de Bases de Datos.

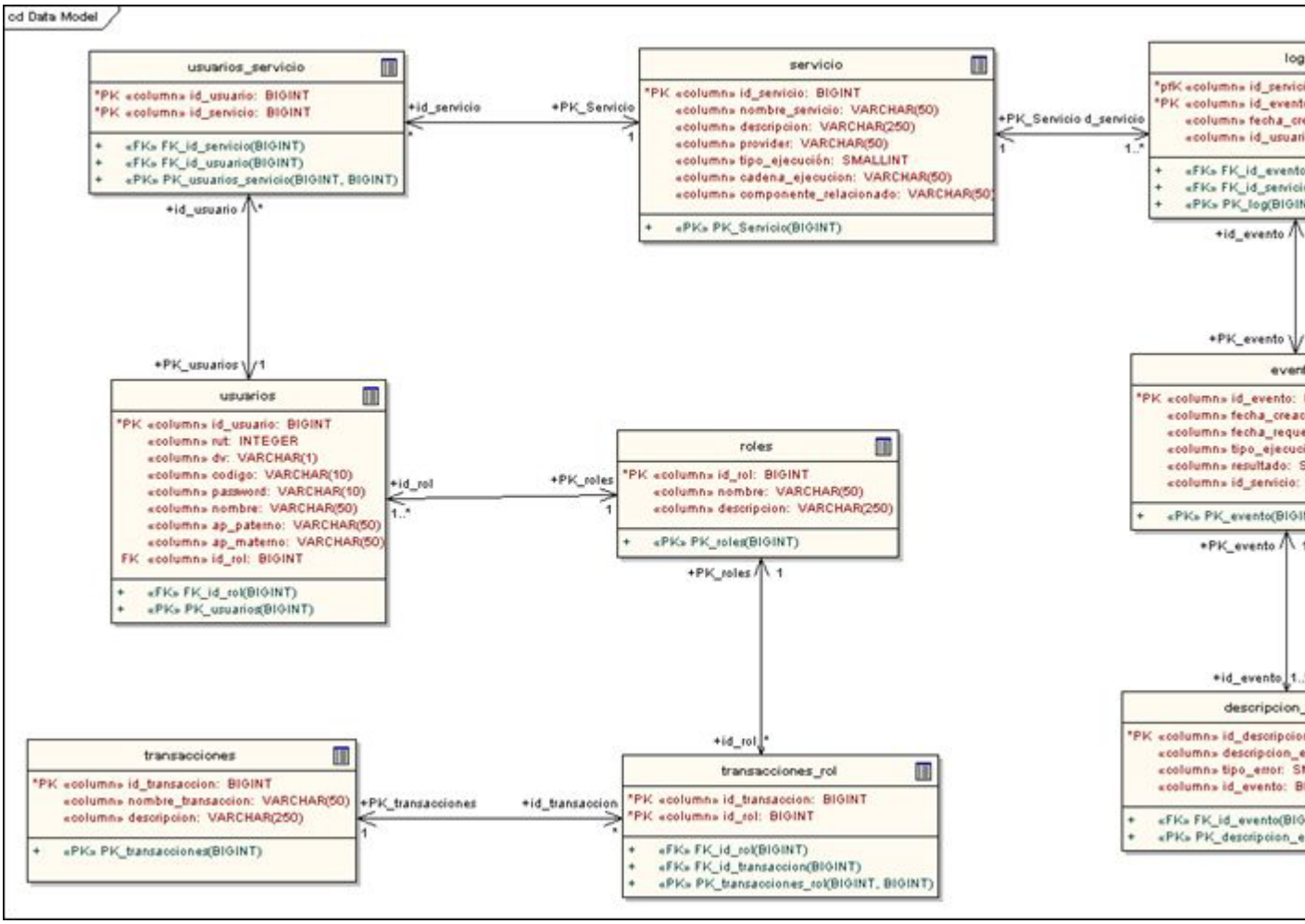

Figura 4-8: Modelo de Base de Datos.

4.6 Implementación de Servicios.

El presente apartado contiene la descripción de de dos servicios requeridos por la Institución Financiera bajo la implementación de tecnología SOA, que corresponde a *obtención de datos de contacto* de un cliente y *gestionar mandatos* asociados al pago de pólizas de seguros.

En términos de negocio, un servicio que proporcione datos de contacto de clientes, favorece la gestión de productos y aumenta la competitividad de la empresa sobre los nichos de mercado objetivo. Al situarnos en el aspecto tecnológico, la implementación de este

servicio implica la integración y homologación de información de clientes y productos de los distintos sistemas operacionales en forma eficiente y eficaz.

El segundo servicio servirá de apoyo a las actividades del departamento operacional de la compañía, integrando el sistema que opera las transacciones que establecen el comportamiento de pago de un cliente, con el sistema que establece un flujo de tareas para los empleados operacionales, permitiendo realizar un seguimiento al ciclo de vida de un producto.

La figura 4-9 representa la arquitectura basada en SOA implementada para ejecutar los servicios requeridos.

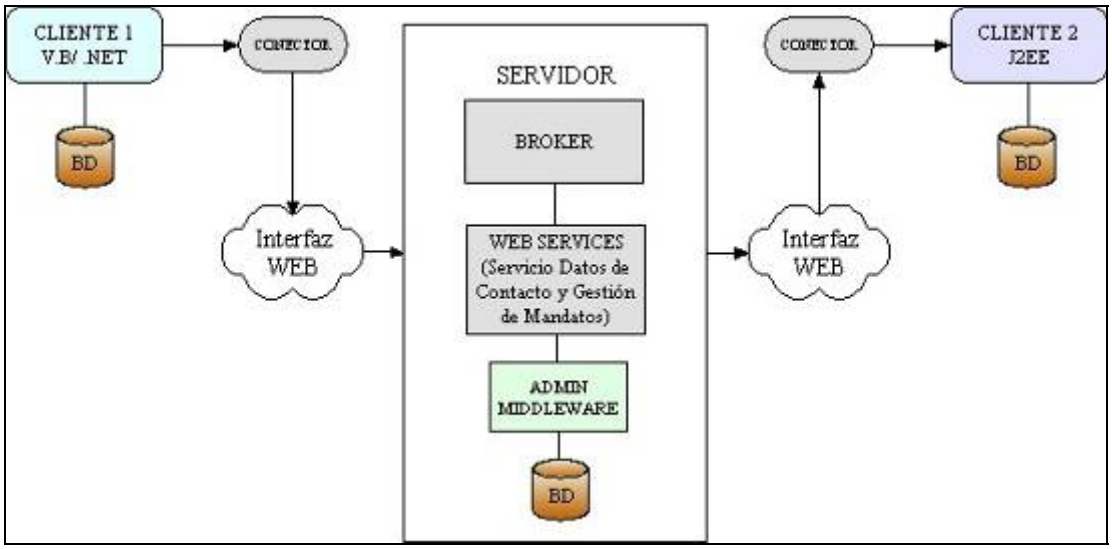

Figura 4-9: Arquitectura de servicios.

4.6.1 Servidor.

El *Servidor* se compone de tres principales elementos; el Broker, los servicios disponibles y una interfaz para administrar el middleware. Las siguientes figuras corresponden a la aplicación que gestiona los servicios implementados.

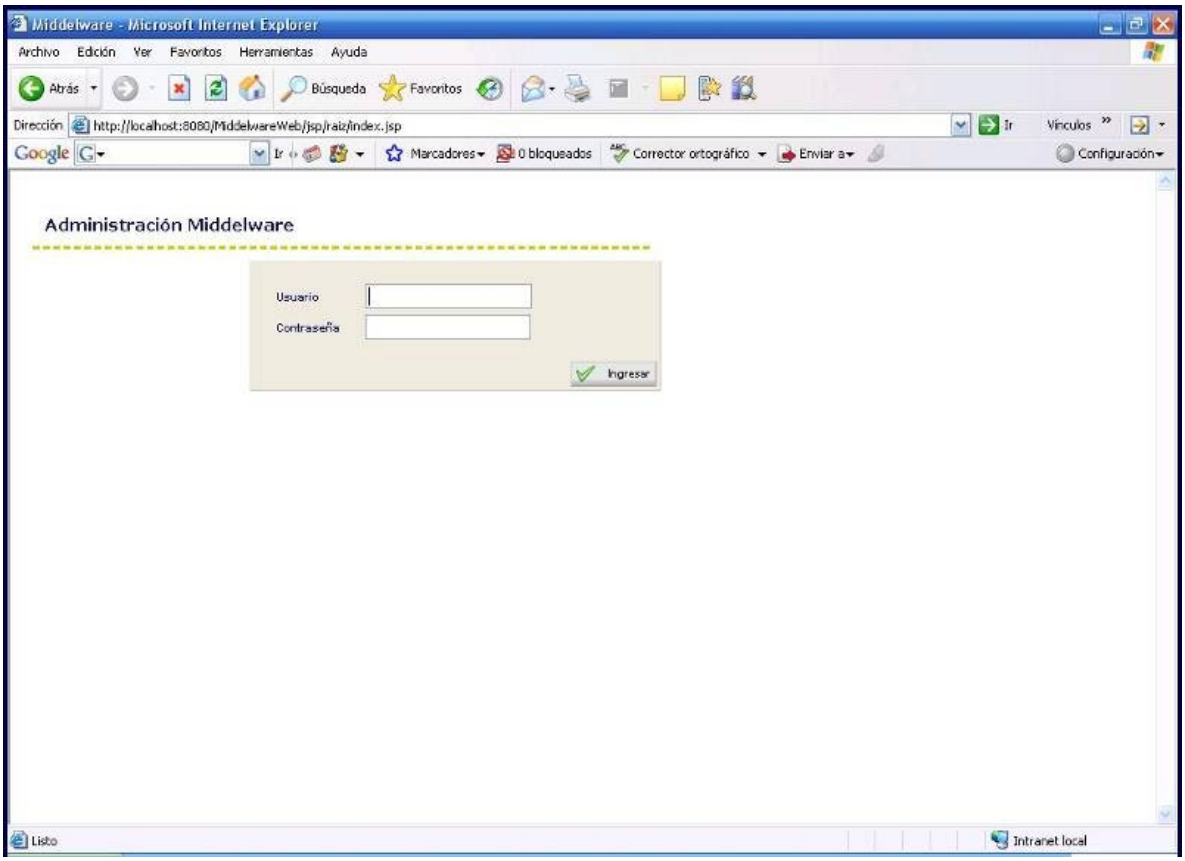

Figura 4-10: Administración del Middleware.

La figura 4-10 muestra la pantalla de inicio con el ingreso a la administración de la configuración del middleware.

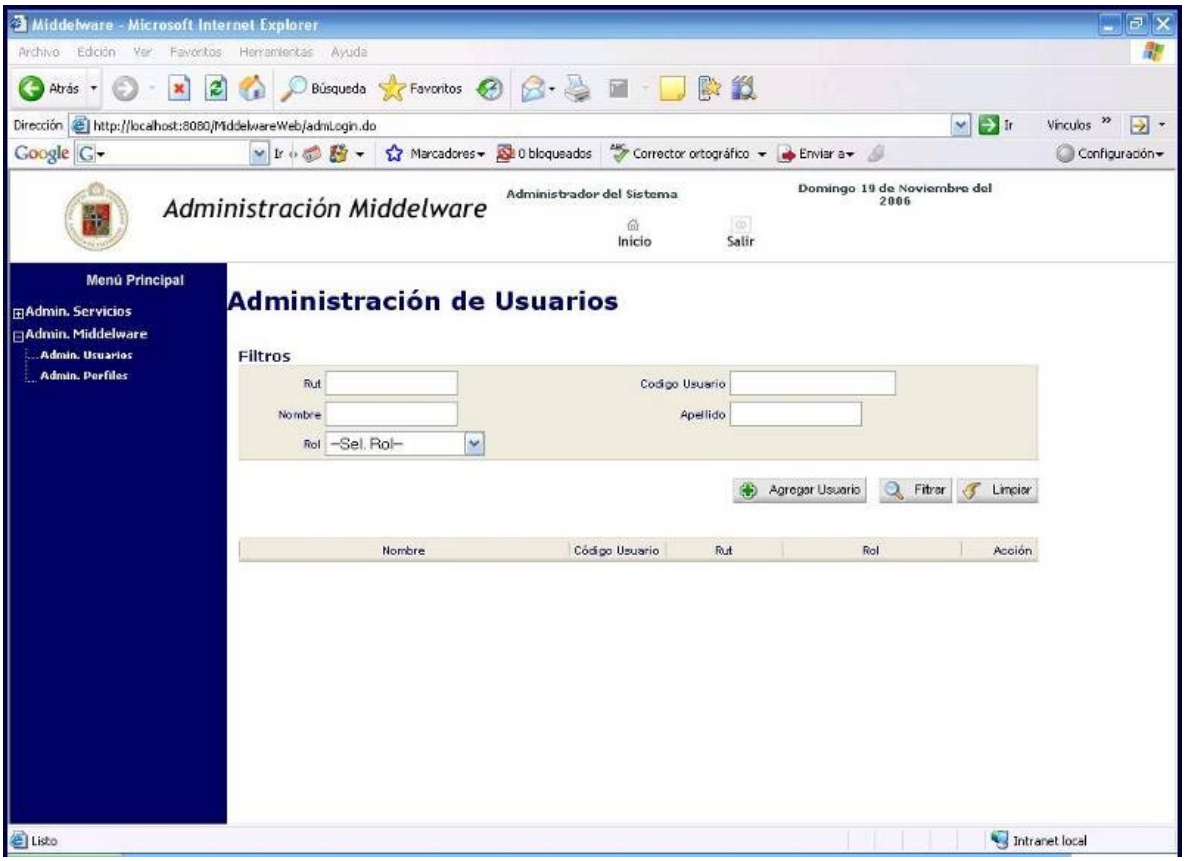

Figura 4-11: Administración del Middleware.

La figura 4-11 muestra la pantalla de ingreso a la administración de usuarios que administran el middleware. Las siguientes dos figuras corresponden a las interfaces para agregar y modificar los datos y permisos de los usuarios administradores.

Figura 4-12: Agregar Usuario.

Figura 4-13: Modificar Usuario.

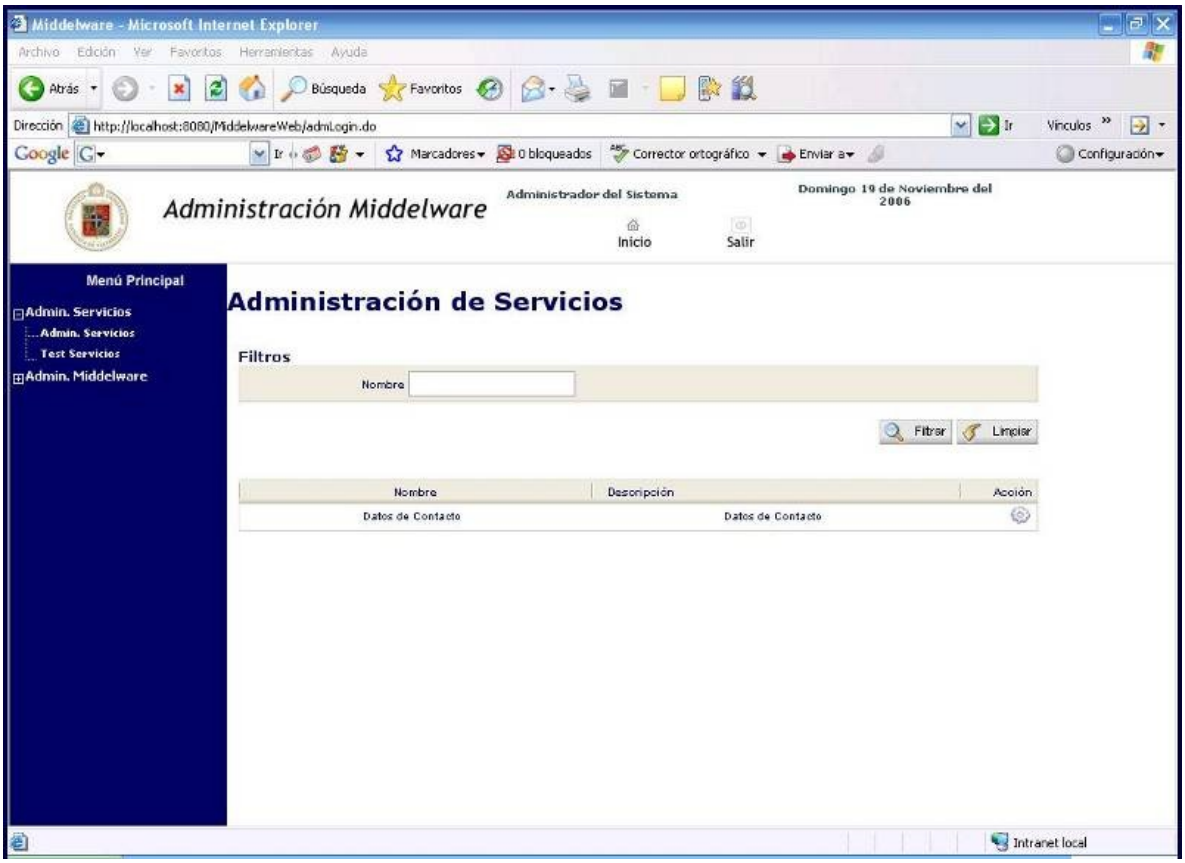

Figura 4-14: Administración de Servicios.

La figura 4-14 muestra la pantalla de Administración de Servicios que provee la vista de los servicios disponibles, además del acceso a la interfaz WSDL, que permite conocer la estructura del servicio y su disponibilidad en línea. La siguiente figura muestra un ejemplo de esta vista:

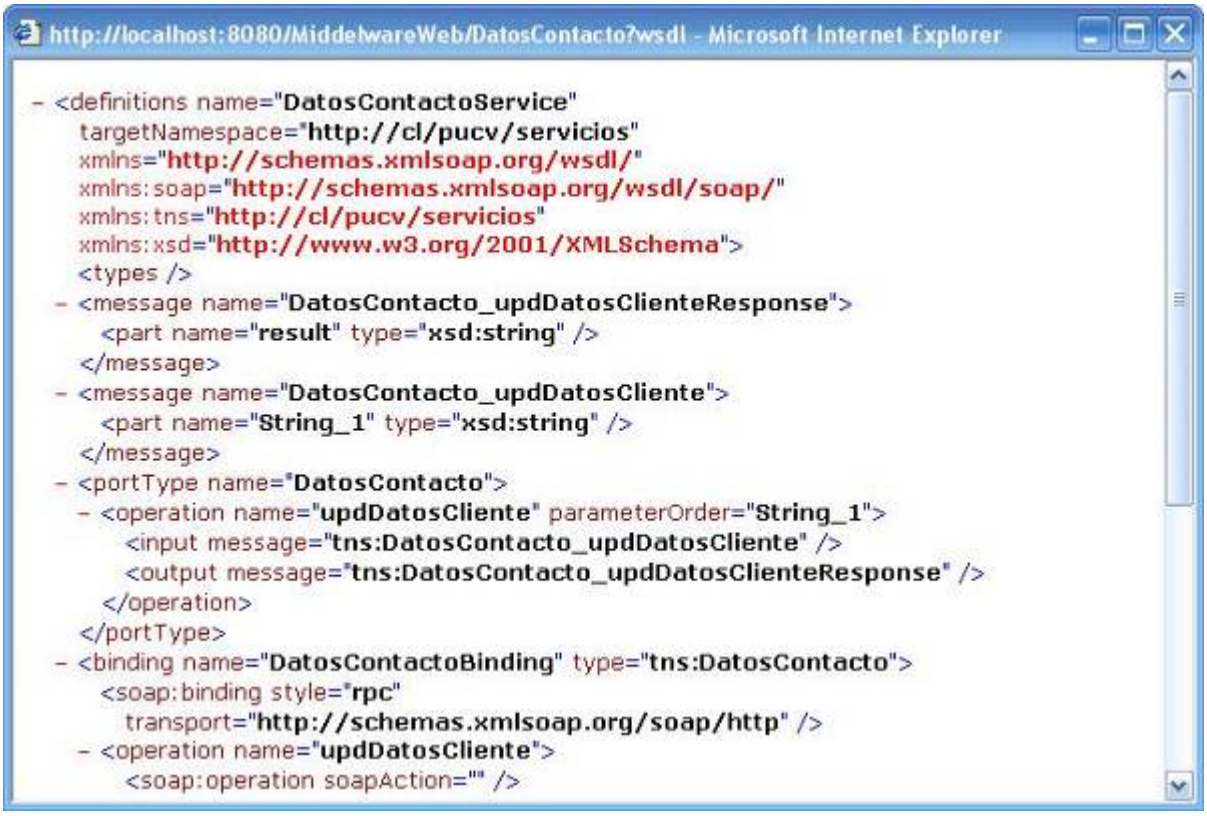

Figura 4-15: Vista de los servicios disponibles del middleware.

4.6.2 Cliente.

Los servicios de prueba se compone de dos clientes, el primero de ellos tiene relación con el ingreso de datos del cliente y el producto asociado a él, desde una aplicación implementada en :NET. La siguiente figura representa la interfaz de prueba del cliente:

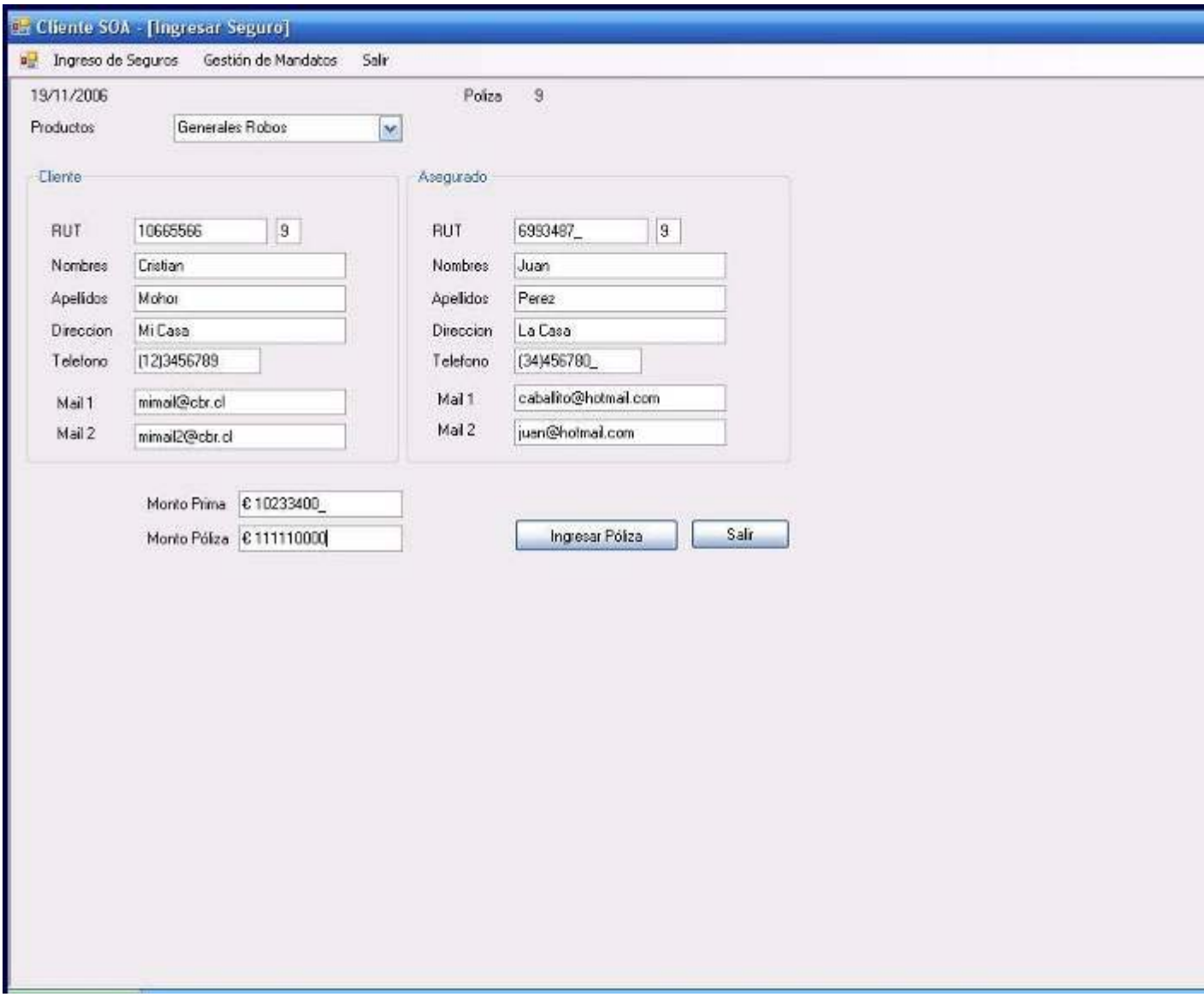

Figura 4-16: Cliente .NET.

El segundo cliente que interactúa con los servicios corresponde a una implementación en J2EE, mostrado en la figura 4-17.

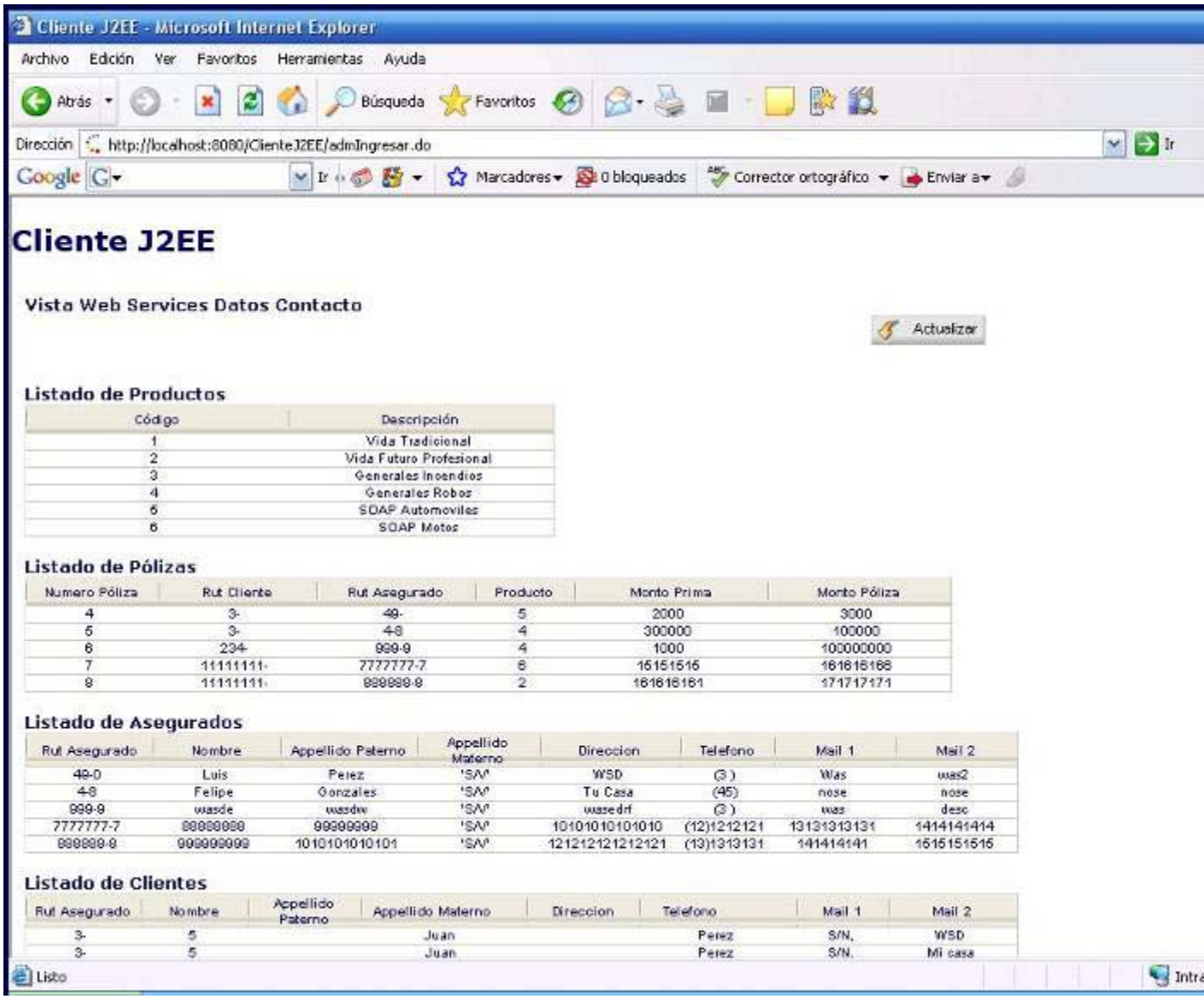

Figura 4-17: Cliente J2EE.

4.6.3 Servicio Datos de Contacto.

El *Servicio de Datos de Contacto* tiene por objetivo potenciar la labor de comunicación de los ejecutivos de venta hacia sus clientes; disminuyendo la tasa de errores en su contacto, posibilitando mejoras referentes a la realización de campañas comerciales, además de mejorar los elementos de difusión y propaganda de los servicios y productos de la compañía. Estas variables serás las que permitirán a los ejecutivos establecer una relación

de confianza con sus clientes, obteniendo una mayor fidelidad y lealtad, reflejados en los resultados de las campañas de ventas.

De esta manera se busca obtener un elemento diferenciador con respecto a las demás compañías en cuanto al servicio que prestan a la atención del cliente.

Para la implementación de este servicio se consideraron tres grandes etapas:

Normalización del ingreso de datos en los sistemas operacionales. Esta etapa incluye la actualización de todos los sistemas operacionales, para lo cual se definieron reglas para el ingreso de información del cliente. De esta manera se establecen estructuras estándares para determinar los datos del cliente.

Un ejemplo de esta actualización es el ingreso de al menos dos teléfonos de contacto y una dirección e-mail, de manera de ampliar las posibilidades de contacto con el cliente.

Normalización y validación de los datos. Esta etapa consiste en realizar una validación de toda la información almacenada comenzando con los datos que ya se tenían, logrando por ejemplo, la estandarización de las direcciones y correo electrónicos de los clientes.

Integración. En esta etapa se integran los datos de contacto a los sistemas. De esta manera, el servicio permite que los cambios realizados en las plataformas operacionales se actualicen dinámicamente en la aplicación CRM de PeopleSoft y viceversa. Además se ingresan automáticamente los nuevos clientes que se registren en el sistema operacional.

4.6.4 Servicio Gestión de Mandatos.

El *Servicio de Gestión de Mandatos* tiene por objetivo mejorar el proceso de negocio al momento de crear la instancia de un nuevo cliente con un producto a través de un mandato de pago el cual es un convenio con el banco para el cargo automático del pago a su cuenta corriente o tarjeta de crédito.

El concepto de mandato corresponde a un contrato con el banco para el cargo de un pago para un producto o varios productos. En este sistema se registra el producto, su comportamiento de pago y su relación contable. Una vez registrado, se extrae del sistema la

información de los libros contables y se establecen los siniestros que cubre el seguro para cada la póliza, finalizando con la generación de las instancias de pago.

La Gestión de Mandatos es una actividad dentro del flujo de negocios que determina una póliza, en la cual el cliente establece un mandato que autoriza a la compañía a generar el pago de la póliza a través de un modo determinado (descuento en su cuenta corriente de un banco, visa, descuento por planilla, etc.). El flujo de creación desde una propuesta o simulación hasta que se transforma en póliza, está establecido en un sistema operacional que integra el producto junto con sus características (cliente, asegurado, coberturas afectas). En otro sistema operacional se establece un flujo de tareas a cargo personal administrativo que genera los pasos a seguir para establecer el convenio de pago, este sistema es *PeopleSoft*, donde existe creado un flujo que permite administrar los estados del mandato.

Resumen del Capítulo.

En este capítulo se mostró el detalle de la implementación de una arquitectura SOA, considerando que existen aplicaciones y su gran necesidad actual es realizar integración entre ellas, de manera que permita mejorar el desempeño de la compañía a nivel operacional y comercial.

La primera parte del capítulo se explicó la situación actual, de manera de determinar las distintas tecnologías y sistemas deseables de comunicar entre sí, luego de esta fase se procede a explicar los requerimientos funcionales y no funcionales que debe suplir la arquitectura planteada para luego dar paso al análisis y diseño del nuevo marco de trabajo.

Finaliza el capitulo con la explicación de los dos servicios de ejemplo que se implementaron como demostración de la arquitectura. El primer servicio, que tiene como objetivo suplir una necesidad comercial, permite establecer la integración entre dos sistemas mejorando la tasa de error de contactos con los clientes unificando las direcciones.

El segundo servicio permite mejorar la administración y gestión de mandatos, los cuales son documentos que el cliente firma para realizar el pago de un servicio con cargo a la cuenta corriente o tarjeta de crédito del cliente. Con esta unificación se permite hacer un

mejor seguimiento de los estados y responsabilidades de los usuarios que ingresan y validan un mandato.

5 Conclusiones

Las grandes compañías invierten gran cantidad de su presupuesto en distintas plataformas tecnológicas, generando soluciones a medida que generan silos de información aisladas, dificultando su integración. Esto provoca que en la organización se genere una arquitectura accidental donde el principal problema está dado por la redundancia de datos y procesos.

Con el objetivo de minimizar estos problemas surgen arquitecturas que comunican diversas plataformas disminuyendo la cantidad de desarrollo para realizar integración de las aplicaciones. Dentro de este contexto surge el concepto de SOA (Arquitectura Orientada a Servicios) que permite realizar integración entre sistemas a bajo costo debido al mínimo acoplamiento de sus componentes.

SOA es una arquitectura que pone orden dentro de los desarrollos que existen y los que vienen, dando una alta escalabilidad a los sistemas transaccionales de la compañía, puesto que se basa en el concepto de servicio. Estos poseen un bajo acoplamiento entre sí, donde se definen funcionalidades específicas y genéricas a todo el negocio, otorgando un alto grado de integración transversal a la organización.

Por otro lado la tecnología Web Services, ampliamente aceptada dentro del mundo informático, permite la generación de funcionalidades que se adaptan a los distintos requerimientos del negocio, siendo de especial interés la generación de servicios que representan funcionalidades internas de la compañía y servicios que permiten comunicarse con otras compañías en modalidad B2B.

SOA provee un marco que administra y ejecuta estos servicios, siendo posible la utilización de funcionalidades que se encuentren en distintas plataformas tecnológicas, como por ejemplo un sistema J2EE con un sistema .NET, dando un estándar de comunicación a través de un lenguaje estándar, como lo puede ser XML.

Establecido el marco teórico de SOA, la aplicación de esta tecnología se pudo llevar a la práctica a través de la implementación de dos servicios que cubren necesidades comerciales y operativas de una institución financiera escogida como caso de estudio.

Para ello, se realizó un análisis de la situación actual, el diseño y posterior implementación de una arquitectura basada en SOA que comunicaba los servicios de una plataforma cliente de tecnología .NET hacia otra que utilizaba J2EE.

De esta manera, uno de los servicios realiza la integración y homologación de información de los clientes asociados a sus productos. El segundo servicio por su parte, permite gestionar los mandatos de las pólizas que contratan los clientes. Para ello, integra el sistema que administra las transacciones que establecen el comportamiento de pago de un cliente, con el sistema que establece un flujo de tareas para los empleados operacionales, permitiendo realizar un seguimiento al ciclo de vida de un producto.

Finalizada la última etapa descrita en los objetivos específicos, se llega a la conclusión que el análisis y el diseño planteado para el caso práctico están dentro de los marcos de implementación de SOA, y cumple con el Objetivo General planteado al inicio del Proyecto.

6 Referencias

[1] Lawrence Pfleeger, Shari. “Ingeniería de Software. Teoría y Práctica”. Editorial Prentice-Hall, 2002.

[2] Pressman, Roger. “Ingeniería de Software. Un enfoque práctico”. Editorial McGraw Hill, 2001.

[3] Dewayne E. Perry y Alexander L. Wolf. “Foundations for the study of software architecture”. ACM SIGSOFT Software Engineering Notes, 17(4), pp. 40–52, Octubre de 1992.

[4] Dijkstra, Edsger. “The Structure of the THE Multiprogramming system.” Communications of the ACM, 26(1), pp. 49-52, Enero de 1983.

[5] Wirth, Niklaus. “Program development by stepwise refinement”, Communications of the ACM, 14(4), pp. 221-227, Abril de 1971.

[6] DeRemer, Frank y Kron, Hans. “Programming-in-the-large versus programming-in-the-small”. IEEE Transaction in Software Engineering, 2, pp. 80-86, 1976.

[7] Brooks Jr, Frederick. “The mythical man-month”. Reading, Addison-Wesley, 1975.

[8] Parnas, Davis. “On the Criteria for Decomposing Systems into Modules.” Communications of the ACM 15(12), pp. 1053-1058, Diciembre de 1972.

[9] Parnas, David. “On a ‘Buzzword’: Hierarchical Structure”, Programming Methodology, pp. 335-342. Berlin, Springer-Verlag, 1978.

[10] Parnas, David. “On the Design and Development of Program Families.” IEEE Transactions on Software Engineering SE-2, 1, pp. 1-9, Marzo de 1976.

[11] Clements, Paul y Northrop, Linda. “Software architecture: An executive overview”. Technical Report, CMU/SEI-96-TR-003, ESC-TR-96-003. Febrero de 1996.

[12] Shaw, Mary. “Abstraction Techniques in Modern Programming Languages”. IEEE Software, Octubre, pp. 10-26, 1984.

[13] Shaw, Mary. “Large Scale Systems Require Higher- Level Abstraction”. Proceedings of Fifth International Workshop on Software Specification and Design, IEEE Computer Society, pp. 143-146, 1989.

[14] WWISA. “Philosophy”. Worldwide Institute of Software Architects, http://www.wwisa.org , 1999.

[15] Baragry, Jason y Reed, Carl. “Why We Need a Different View of Software Architecture”. The Working IEEE/IFIP Conference on Software Architecture (WICSA), Amsterdam, 2001.

[16] Fielding, Roy Thomas. “Architectural styles and the design of network-based software architectures”. Tesis doctoral, University of California, Irvine, 2000.

[17] SOA. “Service Oriented Architecture”, http://www.service-architecture.com

[18] World Wide Web Consortium (W3C). “Simple Object Acces Protocol (SOAP)”. http://www.w3.org/TR/SOAP y http://develop.com/soap

[19] World Wide Web Consortium (W3C). “Web Services Description Language (WSDL)”. http://www.w3.org/TR/wsdl

[20] Business Process Execution Language (BPEL). http://www.service-architecture.com

[21] Java Remote Method Invocation (Java RMI). http://java.sun.com/products/jdk/rmi/index.jsp

[22] Common Object-Request Broker Arquitectura (CORBA). www.corba.org

[23] Universal Description Discovery and Integration (UDDI). http://www.uddi.org , http://uddi.microsoft.com/ y http://www-3.ibm.com/services/uddi/

[24] Historia de los Web Services Sitio DesarrolloWeb, http://www.desarrolloweb.com/articulos/1883.php?manual=61#manuales

[25] Sitio web IBM http://www.ibm.com/ar/desarrolladores/conozca.phtml#1

[26] Librerías técnicas sitio oficial IBM http://www-128.ibm.com/developerworks/views/webservices/library.jsp?S_TACT=103AMW04&S_CMP=sswstile

[27] Sitio web official Microsoft http://msdn.microsoft.com/webservices/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/webservbasics.asp

[28] eBA e-Soluciones de Banda Ancha whitepaper Web Services, Telefonica, 2003

[29] Artículos técnicos Grupo Danysoft: Introducción a Web Services con herramientas de desarrollo Microsoft Por Oscar González Moreno Equipo Grupo Danysoft, abril de 2002

[30] Web Services.Introducción y Escenarios para su Uso.Moisés Daniel Díaz Toledano.

[31] Sitio web official Microsoft http://msdn.microsoft.com/webservices/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/webservbasics.asp

[32] Eric Newcomer, Greg Lomow, “Understanding SOA with Web Services”, December 14, 2004

[33] Dirk Krafzig, Karl Banke, Dirk Slama, “Enterprise SOA: Service-Oriented Architecture Best Practices”, November 09, 2004.

[34] Whitepaper from Cape Clear Software Inc., Principals of SOA Design, 2004 Cape Clear Software.

[35] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pal Krogdahl, Min Luo, Tony Newling, “Patterns: Service-Oriented Architecture and Web Services”, Redbooks IBM Corp. 2004.

7 Glosario

.NET: Es un proyecto de Microsoft para crear plataformas de desarrollo de software con énfasis en transparencia de redes, independencia de plataforma y que permite un rápido desarrollo de aplicaciones.

B2B (Business to Business): Se refiere al comercio electrónico entre empresas. Se basa en la interacción de empresas por medio de Internet. Puede incluir intercambios de información, plataformas de subastas y mercados de negocios.

BPEL (BUSINESS PROCESS EXECUTION LANGUAGE): Es un lenguaje que permite la definición de las interacciones entre las diferentes entidades que forman parte del proceso de negocio. Mediante una serie de tags XML se identifican a los diferentes participantes del proceso así como sus interacciones. Cada uno de estos participantes, así como el proceso que los engloba, es visto como un servicio web.

CORBA (Common Object Request Broker Architecture): Es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

DCOM (Distributed Component Object Model): Es una tecnología propietaria de Microsoft para desarrollar componentes software distribuidos sobre varios computadores que se comunican entre sí.

EJB (Enterprise JavaBeans): Son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE de Sun Microsystems. Su objetivo es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, etc.) para centrarse en el desarrollo de la lógica de negocio en sí.

ERP (Enterprise Resource Planning): Son sistemas integrales de gestión para la empresa. Se caracterizan por estar compuestos por diferentes partes integradas en una única

aplicación. Estas partes son de diferente uso, por ejemplo: producción, ventas, compras, logística, contabilidad, gestión de proyectos, etc.

FTP (File Transfer Protocol): Es un protocolo de conexión de un sitio Internet para cargar y descargar ficheros.

IDL (Interface Definition Language): Corresponde a un lenguaje de especificación de interfaces que se usa como parte de la tecnología CORBA. Ofrece la sintaxis necesaria para definir los métodos que queremos invocar remotamente.

J2EE (Java 2 Enterprise Edition): Es la edición empresarial del paquete Java creada y distribuida por Sun Microsystems. Comprenden un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales.

JMS (API de Servicios de Mensajería de Java): es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma de Java 2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona.

MIDDLEWARE: Es un software de conectividad que permite ofrecer un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida que se sitúa entre la capa de aplicaciones y las capas inferiores (sistema operativo y red).

MSMQ: Es una herramienta de Microsoft para realizar el manejo de mensajería.

REST: Modelo de Arquitectura de INTERNET que establece en el año 2000 Roy Fielding, imponiendo definitivamente el tema de las tecnologías de Internet y los modelos orientados a servicios y recursos en el centro de las preocupaciones de la disciplina.

RMI (Java Remote Method Invocation): Es un mecanismo del lenguaje Java que permite a un procedimiento (método, clase, aplicación o como guste llamarlo) poder ser invocado remotamente.

ROI (Return of Investmentes): El beneficio que se obtiene por cada unidad monetaria invertida en tecnología durante un periodo de tiempo. Suele utilizarse para analizar la viabilidad de un proyecto y medir su éxito. Su medida es un número relacionado con el ratio Coste/Beneficio.

RPC (Remote Procedure Call): Es un protocolo que permite a un programa ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

SCM (Supply Chain Management): es el nombre con el que se conoce a las soluciones que se utilizan para suministrar los procesos y valor de un negocio en todos los rincones de la empresa extendida, desde el proveeedor de tu proveedor hasta el cliente de tu cliente. SCM utiliza conceptos e-business y tecnología Web para administrar procesos más allá de la organización.

SERVLETS: Es un objeto que se ejecuta en un servidor o contenedor J2EE, fue especialmente diseñado para ofrecer contenido dinámico desde un servidor Web, generalmente es HTML.

SLA (Service Level Agreement): Es un protocolo plasmado normalmente en un documento de carácter legal por el que una compañía que presta un servicio a otra se compromete a prestar el mismo bajo unas determinadas condiciones y con unas prestaciones mínimas.

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

SOA (Service-oriented architecture): Es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

SOAP (Simple Object Access Protocol): Es un protocolo estándar creado por Microsoft, IBM y otros, está actualmente bajo el auspicio de la W3C que define cómo dos objetos en

diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos utilizados en los servicios Web.

WEB SERVICES: Son una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferente y ejecutada sobre cualquier plataforma pueden utilizar los servicios Web para intercambiar datos en una red de computadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos.

WSDL (Web Services Description Language): Es un formato XML que describe los servicios de red como un conjunto de puntos finales que procesan mensajes contenedores de información orientada tanto a documentos como a procedimientos.

XML (Extensible Markup Language): Es un lenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

Anexo A: Análisis Caso Práctico

A continuación se presenta el análisis completo para la Arquitectura planteada como solución para la institución financiera.

A.1 Requerimientos Caso Práctico.

Dada la situación actual y en conjunto con la investigación realizada, el caso práctico se enfoca a los siguientes requerimientos para montar la nueva Arquitectura Orientada a Servicios. Estos requerimientos están divididos en dos categorías, Requerimientos Funcionales y Requerimientos No Funcionales, en la primera se entrega un listado que determina las funcionalidades que debe tener la nueva Arquitectura. Para la segunda categoría se plantea los requerimientos que no son funcionalidades del sistema y que son relevantes para el mismo.

A.1.1 Requerimientos Funcionales.

A continuación se presenta los Requerimientos Funcionales, con los que la nueva arquitectura debe cumplir al finalizar el proyecto.

Código	Nombre	Descripción
RF-1.1	Administrar Servicios.	Permite ingresar, modificar y eliminar los servicios que se encuentran disponibles en el catálogo de servicios.
RF-1.2	Ejecutar Servicios.	Se refiere a la ejecución de Servicios, ya sea de prueba al crear uno nuevo o en tiempo de ejecución.
RF-1.3	Generar un Bus de Comunicación.	A través de mensajería se permite la comunicación con la nueva arquitectura, donde se recibe un mensaje de petición y la respuesta va en un mensaje.
RF-1.4	Monitoreo	Permite monitorear la disponibilidad de mensajes, bases de datos o servicios.
RF-1.5	Generar Reportes	A partir de la funcionalidad de monitoreo se plantea la necesidad de generar un módulo de reportes que permita visualizar la disponibilidad y el nivel de fallas que ocurren en la arquitectura.
RF-1.6	Administrar Usuarios	Esta funcionalidad permite la administrar los usuarios que van a interactuar con los servicios y con la aplicación de administración.
RF-1.7	Consolidar Datos de Clientes de Póliza	Esta funcionalidad permite generar un servicio que consolide en los sistemas propietarios la información relativa a un cliente de una póliza.

RF-1.8	Administrar Gestión de Mandatos	Servicio que realiza la actualización de los mandatos en los sistemas propietarios.

Tabla A-1: Descripción Requerimientos Funcionales.

A.1.2 Requerimientos No funcionales.

El objetivo de plantear los requerimientos no funcionales corresponde a condiciones que el sistema debe cumplir para dar valor y satisfacer las necesidades de los actores. Es de esta forma que los requerimientos no funcionales corresponden a características que debe tener el sistema para cumplir de manera eficiente y eficaz los requerimientos funcionales. A continuación se detallan los requerimientos no funcionales que debe cumplir la arquitectura:

Requerimientos de seguridad.

El sistema debe tener perfilamiento a nivel de administración de los servicios. De esta manera se permite que sólo usuarios autorizados puedan interactuar con los servicios. A nivel de ejecución de servicios estos también debe poseer perfilamiento, donde se permitirá la realización de la llamada a través de usuarios plenamente identificados.

Requerimientos de Portabilidad.

La consola de administración de los servicios, debe ser diseñada de manera que se pueda tener acceso desde cualquier computador que posea los mínimos requerimientos.

Por otro lado la llamada y respuesta entregada por los servicios debe ser implementada en un lenguaje estándar (XML), de manera que los servicios sean independientes a como el aplicativo los llaman.

Requerimientos de Mantención.

La aplicación es paramétrica, lo que permite al usuario con los privilegios necesarios, modificar los parámetros que están siendo utilizados sin necesidad de intervenir el código.

Requerimientos operacionales.

La aplicación debe considerar un interfaz de usuario amistosa y fácil de usar, debe ser intuitiva y de rápido acceso a la funcionalidad deseada. Además debe poseer óptimos tiempos de respuesta ante los requerimientos de los usuarios.

La actualización de los datos modificados por una operación se debe ver reflejada inmediatamente en la aplicación. El sistema es capaz de mitigar inconsistencia de datos producto de operaciones que involucren datos críticos para el normal funcionamiento del sistema.

Se define que para establecer una interacción válida con el Sistema Propietario se deben crear aplicaciones Java (EJB's) capaces de invocar un Procedimiento Almacenado, con conexión vía JDBC, para interactuar con este sistema. Estos servicios están disponibles a través del catalogo de servicios quedando utilizables para su ejecución por cualquier aplicación.

El sistema debe ser creado bajo un servidor de aplicaciones WebSpehere 5.1. La arquitectura es interna a la organización, en esta etapa no se considera comunicación entre otras compañías, aunque dentro del diseño se debe considerar este punto.

Requerimientos escalabilidad.

El sistema se puede ampliar añadiendo más funcionalidad y/o ofreciendo la funcionalidad actual a más usuarios. Se considera que la aplicación es modular y que permite crecer en forma ordenada y de acuerdo a los requerimientos

Requerimientos de acoplamiento.

Los servicios que entrega la aplicación son dependientes entre sí, es decir, los cambios en una función afectan al resto de servicios que utilizan los operadores, esto es a nivel del aplicativo para administrar los servicios. Por definición los servicios tienen bajo acoplamiento y la interacción entre ellos es independiente de la arquitectura.

A.2 Casos de Uso.

Considerando los requerimientos funcionales la construcción de la nueva arquitectura se puede separar en dos grandes capas. La primera referente a la aplicación que debe soportar el ingreso y modificación de los servicios. Por otro lado, se tiene el módulo que soporta todo lo que es la llamada al servicio, su ejecución y registros de eventos. Con motivo de visualizar las funcionalidades relacionadas con los requerimientos funcionales planteados en el punto 4.3.1, se realiza un análisis a nivel de casos de uso, visualizando tanto diagramas como el respectivo formato narrativo.

A.2.1 Caso de Uso: “Contexto”.

Este caso de uso busca reflejar el contexto general de la nueva arquitectura que cubra con los requerimientos. En la Figura 4.2 se puede apreciar el Diagrama de Caso de Uso. Cabe mencionar que existen dos actores que son los que utilizarían las prestaciones de la arquitectura:

1. Administrador.

2. Aplicativo.

Donde el rol del “Administrador”, radica en administrar la configuración de la arquitectura, donde destaca la agregación y parametrización de servicios en conjunto con el manejo de usuarios.

Por otro lado el rol “Aplicativo”, se refiere a la aplicación que finalmente requiere de un servicio y lo llama a través de la arquitectura.

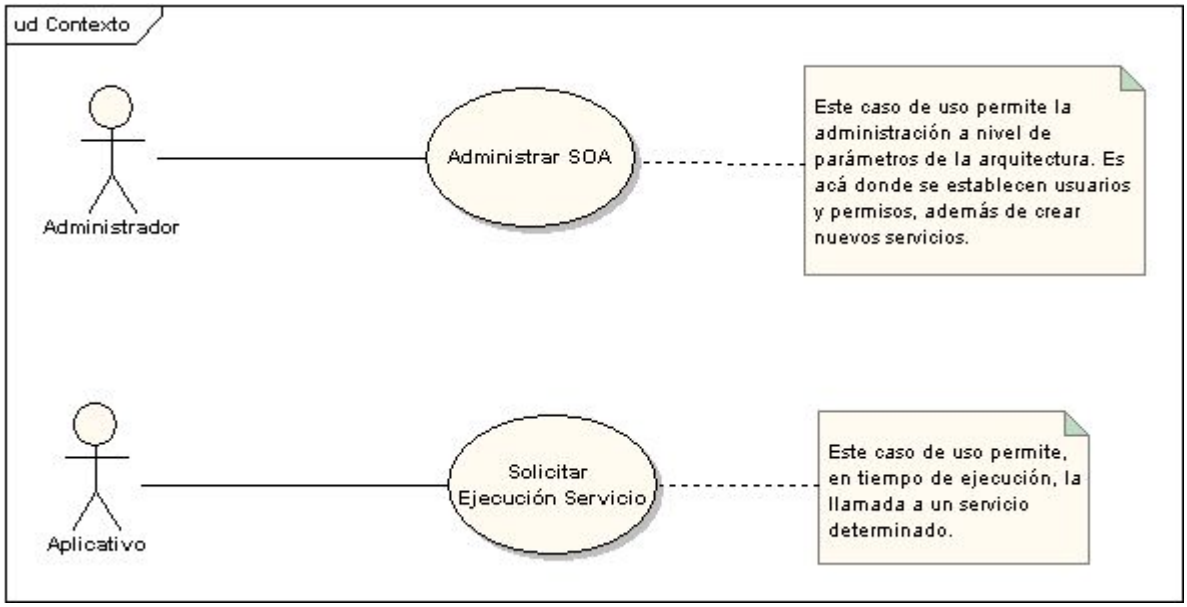

Figura A-1: Diagrama de Caso de Uso: Contexto.

A continuación se presenta el formato narrativo de los casos de uso, presentados en la Figura 4-2A-1, en un formato resumido.

Caso de uso	Administrar SOA.
Descripción	Este caso de uso es genérico, permite la administración de los servicios, usuarios y configuración de la arquitectura.
Actores	Administrador (Iniciador).
Requerimientos	RF-1.1, RF-1.2, RF-1.4, RF-1.5, RF-1.6.
Precondiciones	
Poscondiciones	
Inclusiones	
Extensiones	Administrar Usuario, Monitorear Arquitectura, Administrar Servicios.

Tabla A-1: Formato Narrativo Caso de Uso: Administrar SOA.

Caso de uso	Solicitar Ejecución de Servicio.
Descripción	Este caso de uso permite la llamada de un servicio solicitada por una aplicación.
Actores	Aplicativo (Iniciador).
Requerimientos	RF-1.3.
Precondiciones	
Poscondiciones	
Inclusiones	Recibir Peticiones.
Extensiones	

Tabla A-2: Formato Narrativo Caso de Uso: Solicitar Ejecución de Servicios.

A.2.2 Caso de Uso: “Administrar SOA”.

Este caso de uso cumple con la funcionalidad de configurar parámetros de la arquitectura. En este caso el rol principal viene dado por el “Administrador”, el cual puede agregar usuarios en el sistema, definir sus roles y permisos asociados. Por otro lado es el administrador el que se preocupa de configurar los aplicativos correspondientes al monitoreo y generar los respectivos reportes.

En la Figura 4-32 se puede apreciar el Diagrama de Caso de Uso, en este caso el administrador (iniciador de este caso de uso) tiene la opción de Administrar Servicios, donde es acá donde se puede agregar, modificar, activar/desactivar o ejecutar un servicio. Al agregar un servicio se debe considerar que se deben incluir todos los parámetros de necesarios para su ejecución. Adicionalmente se considera una funcionalidad que permita probar el servicio, ejecutándolo, de manera de validar la correcta parametrización del servicio instalado.

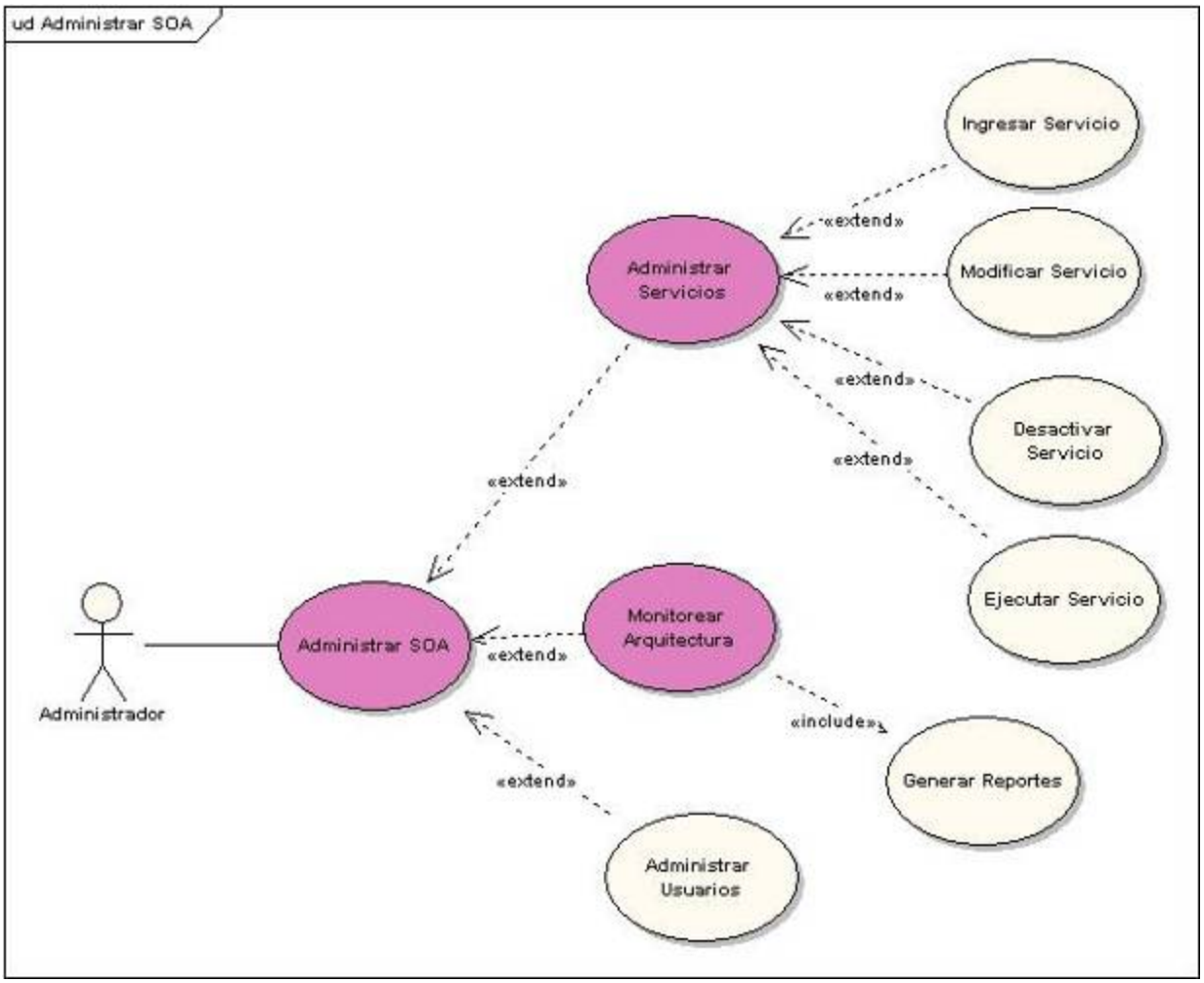

Figura A-1: Diagrama de Caso de Uso: "Administrar SOA".

A continuación se presenta en formatos narrativo la descripción de los casos de uso que se encuentran en la Figura 4-3.

Caso de uso	Administrar Servicios.
Descripción	Este caso de uso es genérico, permite la administración de los servicios. Por administración se entiende el Ingresar, Modificar, Eliminar y Ejecutar un servicio determinado.
Actores	Administrador (Iniciador).

Requerimientos	RF-1.1, RF-1.2.
Precondiciones	
Poscondiciones	
Inclusiones	
Extensiones	Ingresar Servicio, Modificar Servicio, Desactivar Servicio, ejecutar Servicio.

Tabla A-3: Formato Narrativo Caso de Uso: Administrar Servicios.

Caso de uso	Monitorear Arquitectura.
Descripción	Este caso de uso permite monitorear el funcionamiento de la arquitectura, registrando eventos, los cuales pueden ser fallas y peticiones con el objetivo de medir rendimiento y el uso de la nueva plataforma. Esta componente de la arquitectura está hecha en base a demonios los cuales estarán permanentemente ejecutándose para registrar las peticiones y respuestas.
Actores	Administrador (Iniciador).
Requerimientos	RF-1.4.
Precondiciones	
Poscondiciones	
Inclusiones	Generar Reportes.
Extensiones	

Tabla A-4: Formato Narrativo Caso de Uso: Monitorear Arquitectura.

Caso de uso	Administrar Usuarios.
Descripción	Este caso de uso permite ingresar, modificar y eliminar usuarios, estableciendo roles y permisos dentro del sistema solución.
Actores	Administrador (Iniciador).
Requerimientos	RF-1.6.
Precondiciones	
Poscondiciones	
Inclusiones	
Extensiones	

Tabla A-5: Formato Narrativo Caso de Uso: Administrar Usuarios.

Caso de uso	Ingresar Servicios.
Descripción	En este caso de uso se crean los nuevos servicios disponibles dentro de la arquitectura. Se establecen usuarios con permisos, parámetros de entrada y formato de salida, conector a ejecutar y forma de llamada al servicio.
Actores	Administrador (Iniciador).
Requerimientos	RF-1.1.
Precondiciones	
Poscondiciones	Servicio creado en la arquitectura y disponible para su ejecución.
Inclusiones	

Extensiones	

Tabla A-6: Formato Narrativo Caso de Uso: Ingresar Servicios.

Caso de uso	Modificar Servicios.
Descripción	Este caso de uso permite modificar servicios existentes, ya sea la descripción o parámetros de entrada o salida definidos anteriormente..
Actores	Administrador (Iniciador).
Requerimientos	RF-1.1.
Precondiciones	
Poscondiciones	Ser vicio modificado, con nueva configuración.
Inclusiones	
Extensiones	

Tabla A-7: Formato Narrativo Caso de Uso: Modificar Servicios.

Caso de uso	Desactivar Servicios.
Descripción	Este caso de uso eliminar lógicamente un servicio, esta “eliminación” interrumpe la ejecución del servicio afectado. Por otro lado permite también su activación.
Actores	Administrador (Iniciador).
Requerimientos	RF-1.4.

Precondiciones	
Poscondiciones	Servicio eliminado del sistema.
Inclusiones	
Extensiones	

Tabla A-8: Formato Narrativo Caso de Uso: Desactivar Servicios.

Caso de uso	Ejecutar Servicio.
Descripción	Este caso de uso permite ejecutar un servicio de manera de testear su funcionamiento.
Actores	Administrador (Iniciador).
Requerimientos	RF-1.2.
Precondiciones	El servicio debe estar creado en el sistema.
Poscondiciones	
Inclusiones	
Extensiones	

Tabla A-9: Formato Narrativo Caso de Uso: Ejecutar Servicio.

Caso de uso	Generar Reportes.
Descripción	Este caso de uso permite generar reportes a partir de la información registrada por la funcionalidad de monitoreo..
Actores	Administrador (Iniciador).

Requerimientos	RF-1.5.
Precondiciones	La funcionalidad de monitoreo debe estar registrando eventos.
Poscondiciones	
Inclusiones	
Extensiones	

Tabla A-10: Formato Narrativo Caso de Uso: Generar Reportes.

A.2.3 Caso de Uso: “Solicitar Ejecución de Servicio”.

Este caso de uso refleja al core de la arquitectura, donde se puede apreciar el funcionamiento de un flujo de llamada para la ejecución de un servicio por parte de un Aplicativo. En este caso el Aplicativo es el actor principal, el cual mediante un mensaje envía una solicitud de ejecución de un servicio. Este mensaje debe considerar, como medio de seguridad, un usuario y un password válidos para la ejecución.

Luego de consultar y validar el servicio y sus parámetros incluidos en el mensaje, se procede a ejecutar el servicio registrando el evento en un log correspondiente al monitoreo de la arquitectura. A partir de de este evento se retorna un mensaje al aplicativo, el cual recibe la información correspondiente a la ejecución del servicio, en el sentido si fue exitoso o falló en algo.

En la, Figura 4-4 se puede apreciar el Diagrama de Casos de Usos, correspondiente al flujo descrito anteriormente, donde una Aplicación llama a un servicio determinado para cumplir con sus procesos de negocios. A continuación de la Figura 4-4 se presenta en formato narrativo resumido, todos los casos de usos incluidos en el diagrama.

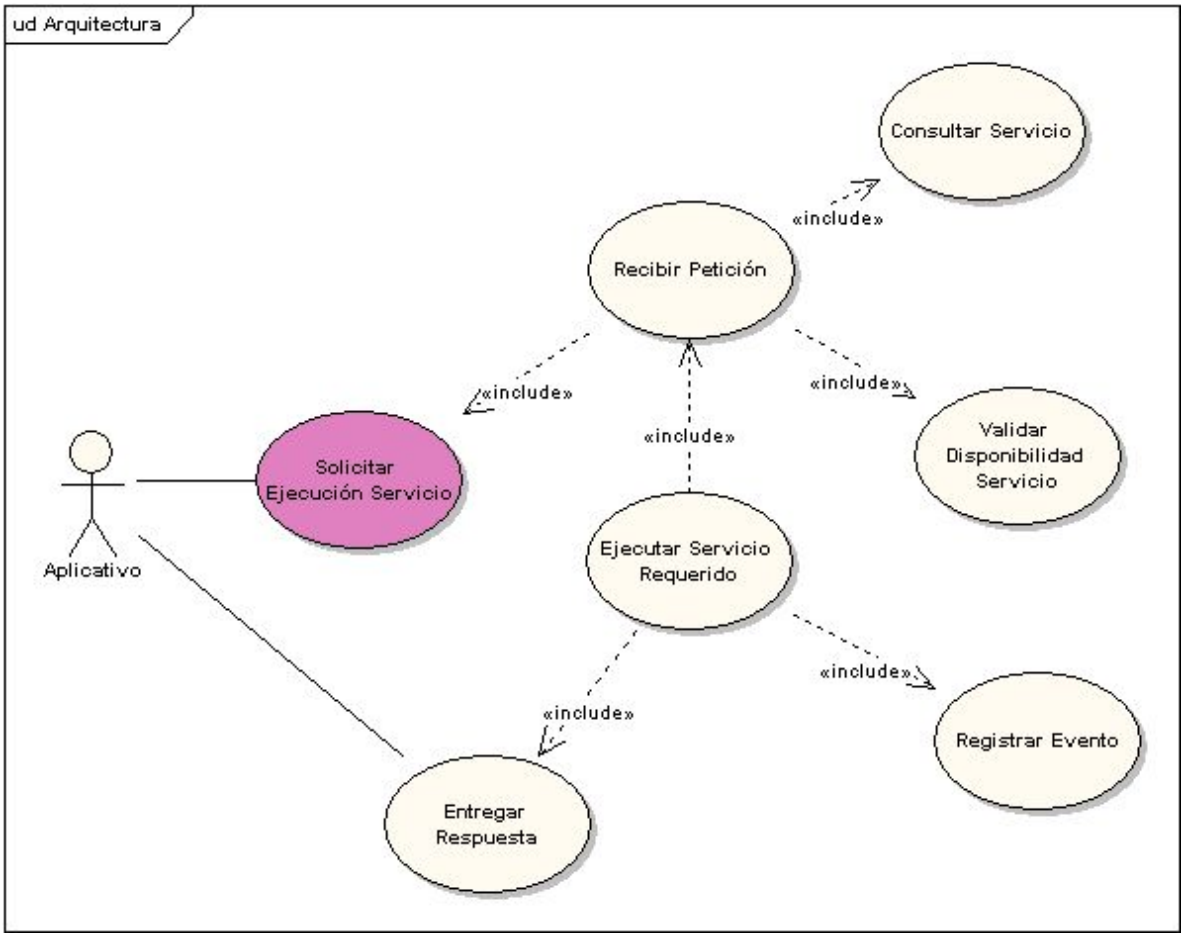

Figura A-2:Diagrama de Caso de Uso: "Solicitar Ejecución Servicio".

Caso de uso	Recibir Petición.
Descripción	Este caso de uso recibe una petición para ejecutar un servicio, esta petición es en forma de mensaje donde viene el servicio a ejecutar, existe un proceso que debe consultar cual es el servicio y como se debe ejecutar.
Actores	Aplicativo (Iniciador).
Requerimientos	RF-1.3.
Precondiciones	El servicio debe estar creado previamente en el sistema.

Poscondiciones	
Inclusiones	Consultar Servicio, Validar Disponibilidad de Servicio
Extensiones	

Tabla A-11: Formato Narrativo Caso de Uso: Recibir Petición.

Caso de uso	Consultar Servicio.
Descripción	Este caso de uso permite consultar sobre la formad de ejecutar un servicio extrayendo parámetros de entrada, parámetros de salida y llamada para su ejecución.
Actores	Aplicativo.
Requerimientos	RF-1.3.
Precondiciones	Debe haber llegado una petición de ejecución de servicios.
Poscondiciones	
Inclusiones	
Extensiones	

Tabla A-12: Formato Narrativo Caso de Uso: Consultar Servicio.

Caso de uso	Validar Disponibilidad del Servicio.
Descripción	Este caso de uso permite validar el estado de un servicio, si está activado o si sus componentes están disponibles.
Actores	Aplicativo.

Requerimientos	RF-1.3.
Precondiciones	El servicio debe de haber sido consultado previamente.
Poscondiciones	
Inclusiones	
Extensiones	

Tabla A-13: Formato Narrativo Caso de Uso: Validar Disponibilidad del Servicio.

Caso de uso	Ejecutar Servicio Requerido.
Descripción	Este caso de uso ejecuta el servicio, tomando la forma de llamarlo con los parámetros de entrada que vienen en el mensaje de petición.
Actores	Aplicativo.
Requerimientos	RF-1.3.
Precondiciones	El servicio debe haber sido validado anteriormente.
Poscondiciones	Petición atendida con éxito o con error
Inclusiones	Recibir Petición, Registrar Evento.
Extensiones	

Tabla A-14: Formato Narrativo Caso de Uso: Ejecutar Servicio Requerido.

Caso de uso	Registrar Evento.
Descripción	Este caso de uso registra en base de datos el estado de la solicitud de servicio, ya sea exitoso o con errores.

Actores	Aplicativo.
Requerimientos	RF-1.3.
Precondiciones	El servicio debe haber sido ejecutado anteriormente.
Poscondiciones	
Inclusiones	
Extensiones	

Tabla A-15: Formato Narrativo Caso de Uso: Registrar Eventos.

Caso de uso	Entregar Respuesta.
Descripción	Entrega la respuesta de la llamada al servicio al aplicativo que realizó la petición de ejecución del servicio. Esta respuesta puede ser un estado o un conjunto de datos dependiendo del tipo de servicio.
Actores	Aplicativo.
Requerimientos	RF-1.3.
Precondiciones	El servicio debe haber sido ejecutado anteriormente.
Poscondiciones	Solicitud atendida por la arquitectura.
Inclusiones	Ejecutar Servicio Requerido.
Extensiones	

Tabla A-16: Formato Narrativo Caso de Uso: Entregar Respuesta.

Anexo B: Diseño Caso Práctico

B.1 Diseño Caso Práctico.

Para enfrentar el diseño requerido para implementar el caso práctico definido anteriormente, se considera los siguientes diagramas:

Diagrama de Secuencia.
Diagrama de Clases.
Modelo de Datos.

Con estos artefactos se espera cubrir el diseño de la arquitectura, de esta forma se espera concluir con las especificaciones necesarias, que cubran el espectro de requerimientos funcionales y no funcionales, para su implementación.

B.1.1 Diagramas de Secuencia.

Los diagramas de secuencia son construidos a partir de lo casos de uso definidos en la etapa de análisis, en este tenor, los casos de uso se separan en dos grandes clasificaciones, los que son propios de los casos de usos visualizados en el nivel de contexto. En una primera instancia se desglosan los respectivos diagramas para el Caso de Uso: “Administrar SOA”, para luego continuar con el Caso de Uso: “Solicitar Ejecución de Servicio”.

B.1.1.1 Administrar SOA.

Para el caso de uso Administrar SOA se tienen los siguientes diagramas de secuencia que explican el flujo existente en las operaciones básicas, para el normal funcionamiento del proceso. En laFigura B-3, se muestra el diagrama de secuencia para el caso de uso Crear Usuario, que si bien es cierto no está detallado en lo casos de uso de la sección 4.4.2 es parte del desglose del Caso de Uso: “Administrar Usuario”, este caso de uso se descompone en cuatro grandes funcionalidades, Agregar Usuario, Modificar Usuario, Eliminar Usuario y Buscar Usuario.

Diagrama de Secuencia: Crear Usuario.

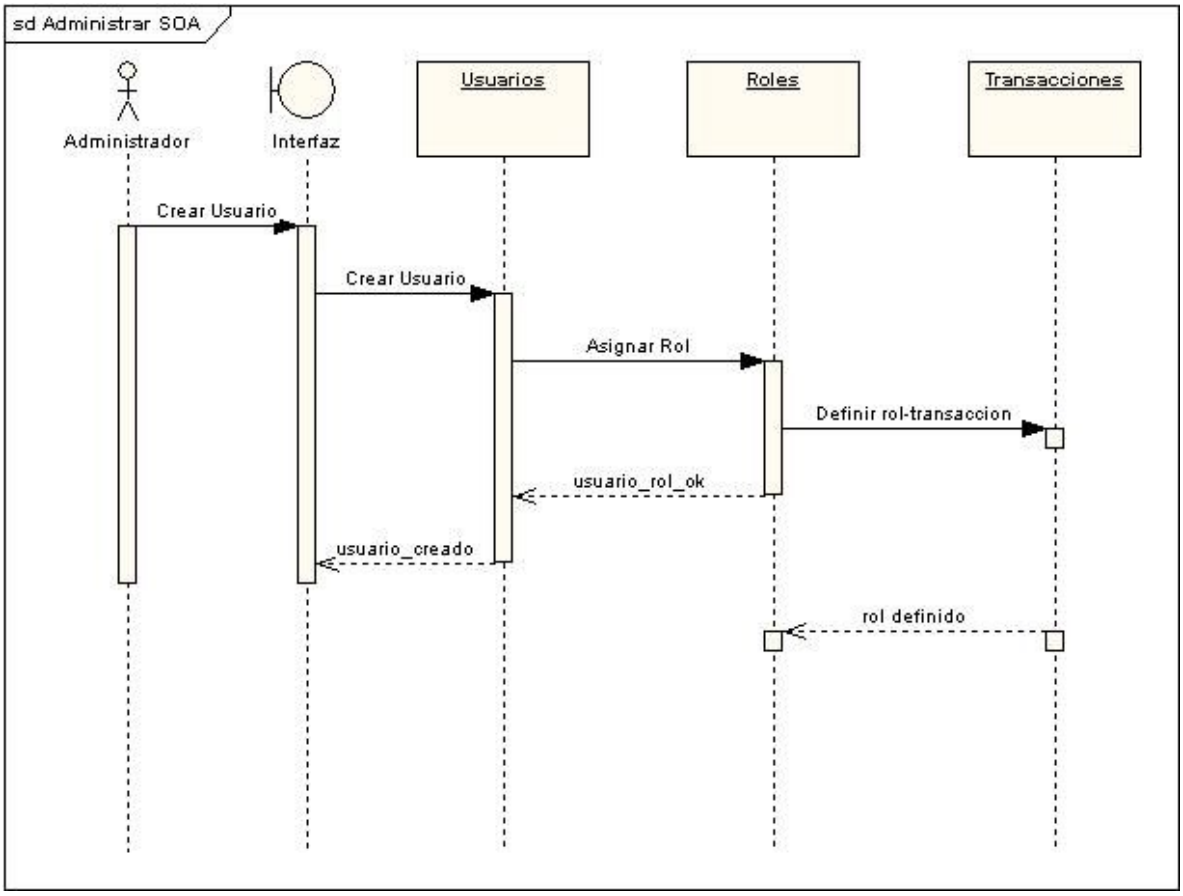

Figura B-3: Diagrama de Secuencia: Crear Usuario.

Diagrama de Secuencia: Modificar Usuario.

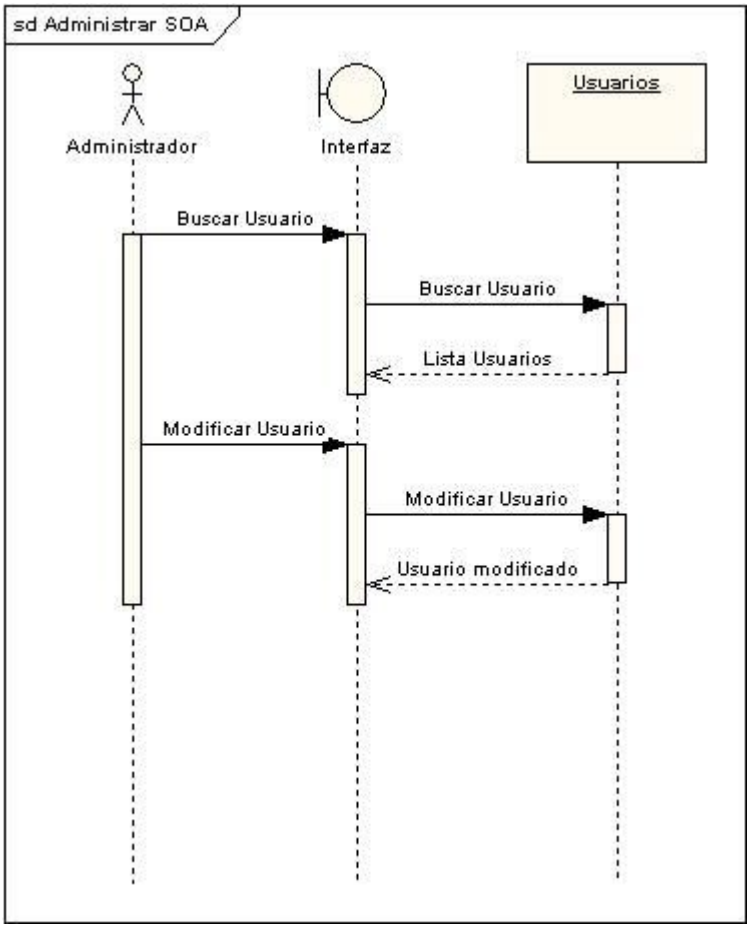

Figura B-4: Diagrama de Secuencia: "Modificar Usuario".

Diagrama de Secuencia: Eliminar Usuario.

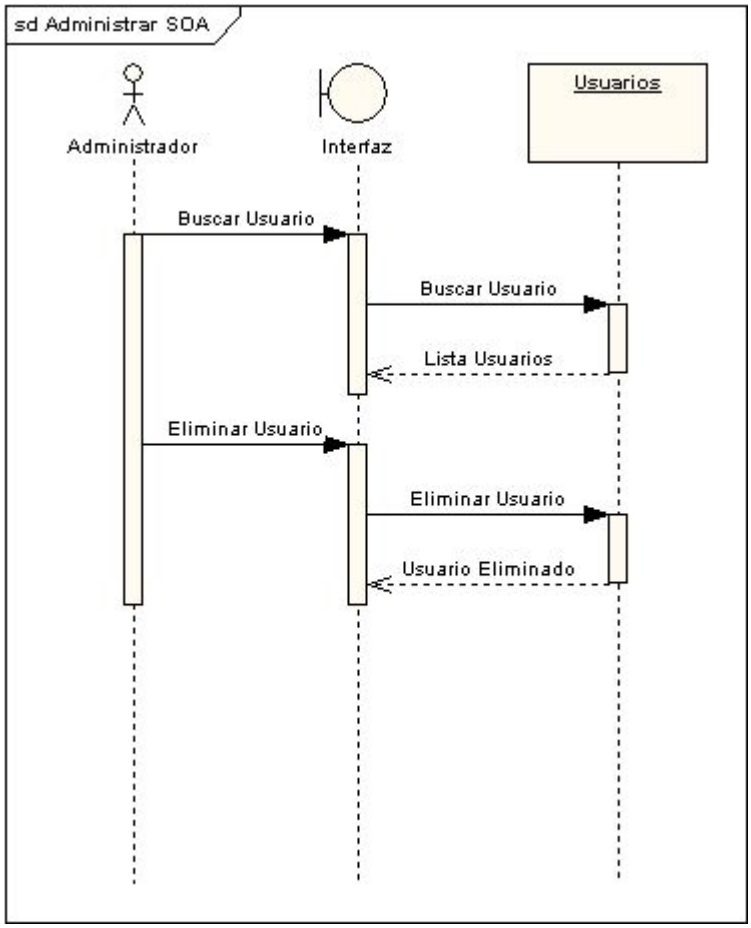

Figura B-5: Diagrama de Secuencia: Eliminar Usuario.

Diagrama de Secuencia: Crear Servicio.

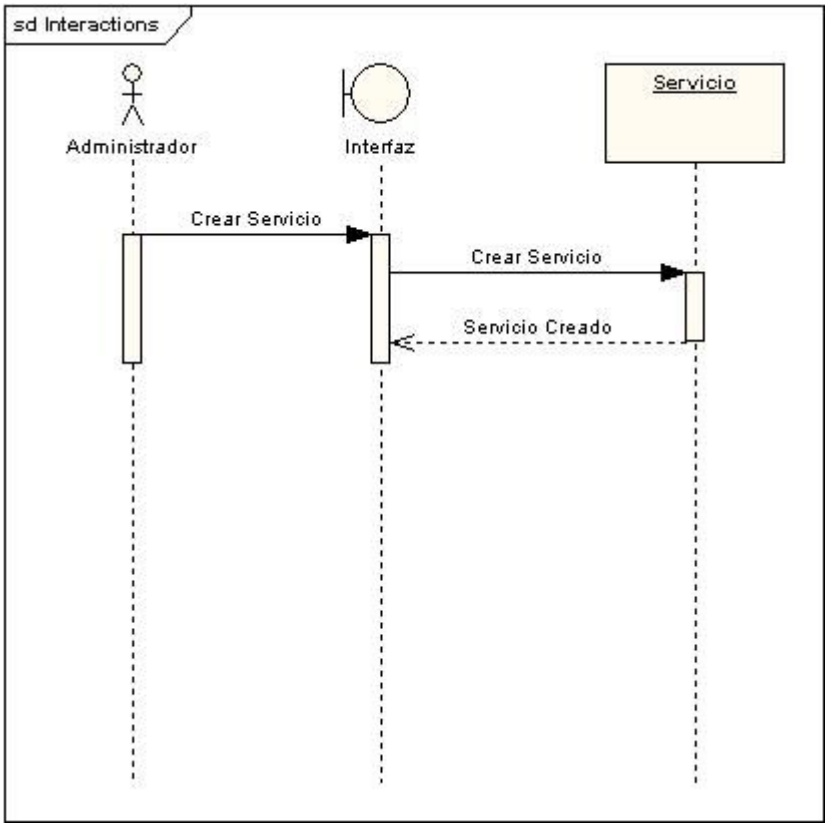

Figura B-6: Diagrama de Secuencia: Crear Servicio.

Diagrama de Secuencia: Ejecutar Servicio.

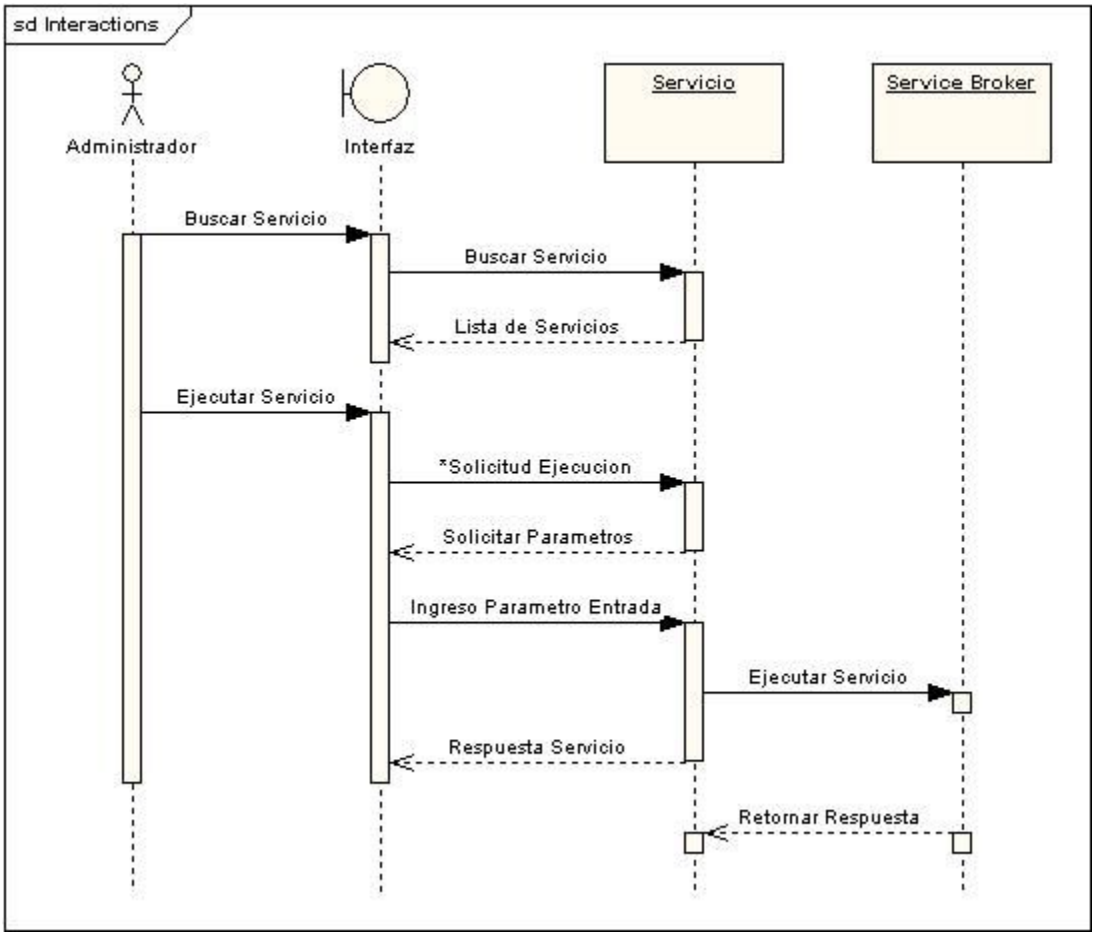

Figura B-7: Diagrama de Secuencia: Ejecutar Servicio.

Diagrama de Secuencia: Activar/Desactivar Servicio.

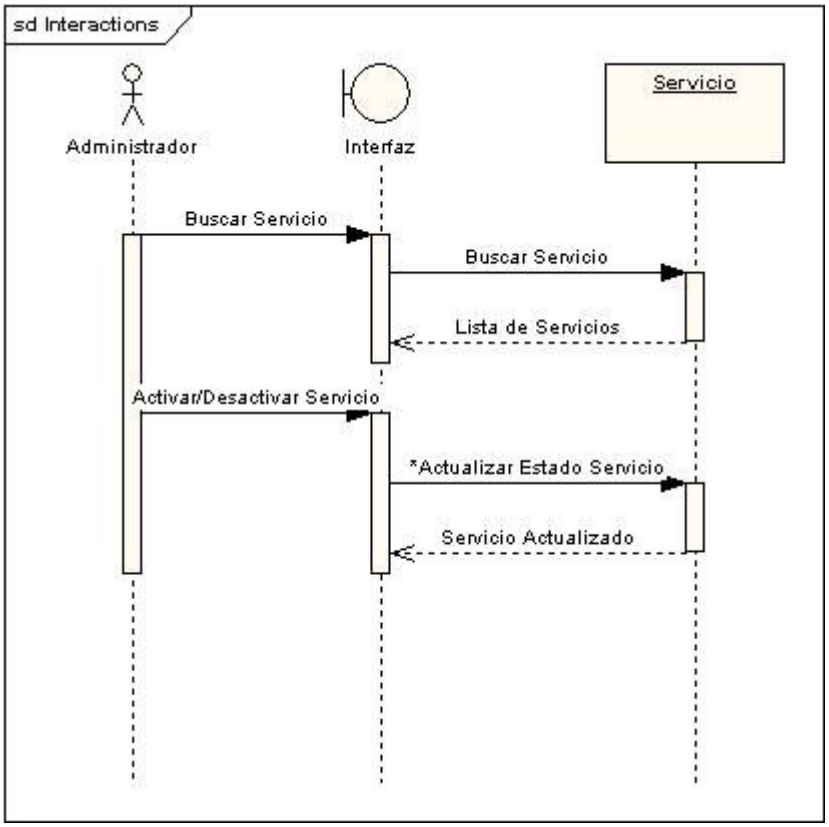

Figura B-8: Diagrama de Secuencia: Activar/Desactivar Servicio.

Diagrama de Secuencia: Modificar Servicio.

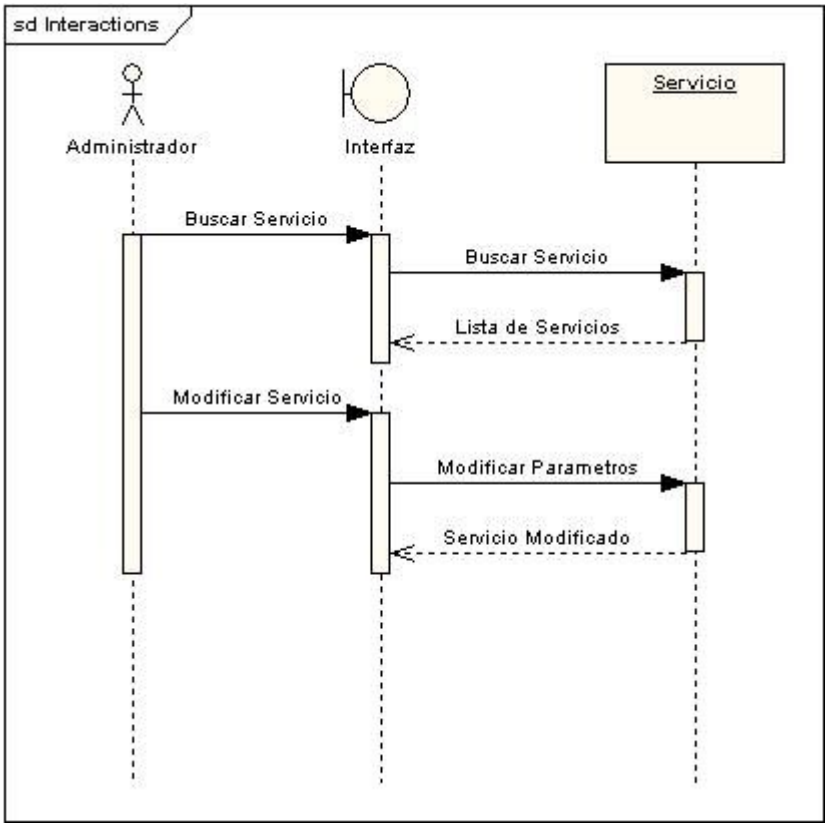

Figura B-9: Diagrama de Secuencia: Modificar Servicio.

Diagrama de Secuencia: Generar Reportes.

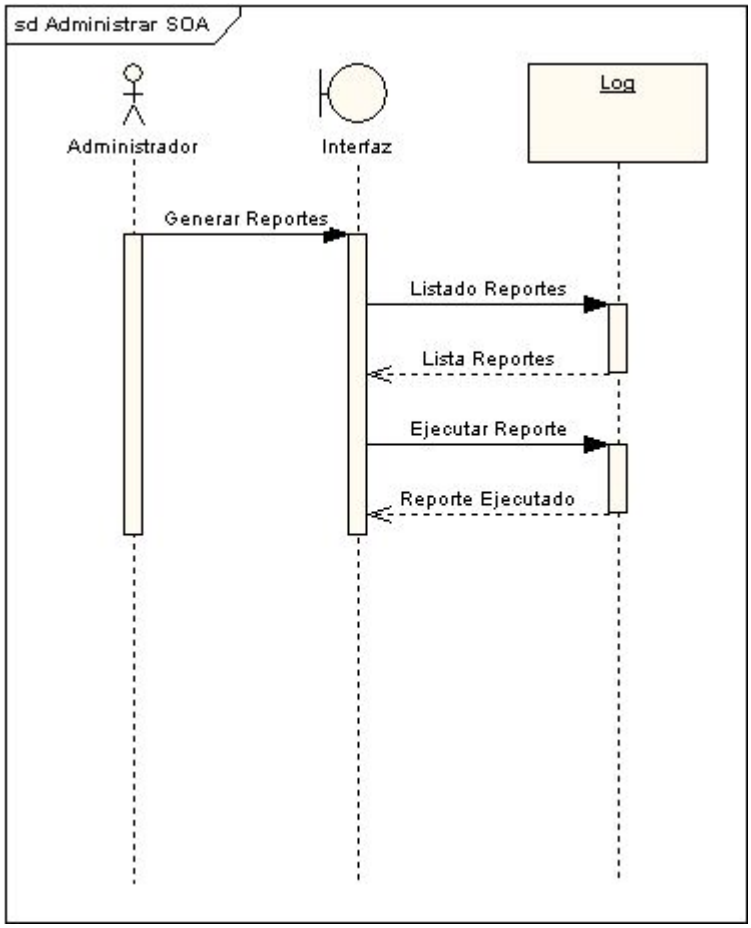

Figura B-10: Diagrama de Secuencia: Generar Reportes.

B.1.1.2 Solicitar Ejecución de Servicio.

En este caso se visualiza el comportamiento de la arquitectura en el momento de ser solicitado, por un aplicativo, la ejecución de un servicio.

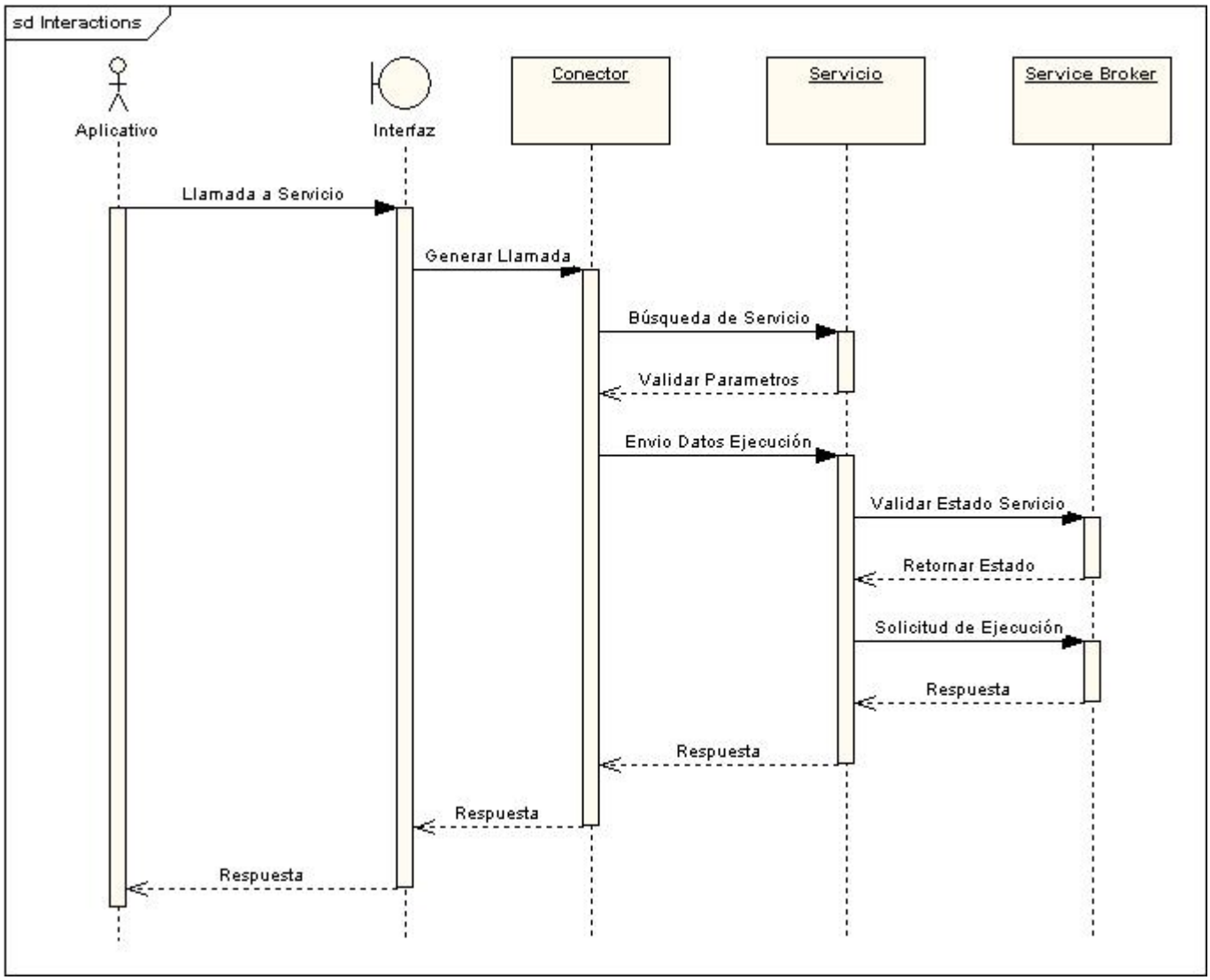

Figura B-11:Diagrama de Secuencia: Solicitar Ejecución de Servicio.

B.1.2 Modelo de Clases.

Para soportar las funcionalidades planteadas para la nueva arquitectura se presenta un diagrama de clases, en un nivel de contexto el cual refleja las clases que se deben considerar para el funcionamiento de la arquitectura. Cabe considerar que el nodo a implementar es el “Service Broker” ya que es en ese nivel donde funciona la arquitectura.

En la Figura 4-6, se puede apreciar el Diagrama de Clases “Nivel 0”, con este diagrama se plantea el funcionamiento a nivel general de la nueva arquitectura. Existe un nodo denominado “Service Consumer”, en el que funcionan todos los aplicativos que requieran

de la ejecución de un servicio determinado, bajo este esquema el Sistema Consumidor envía una petición en forma de mensaje JMS, existe un listener escuchando ese mensaje ante lo cual lo envía al “Bus Arquitectura”, donde ya se sabe que tipo de servicio se debe ejecutar.

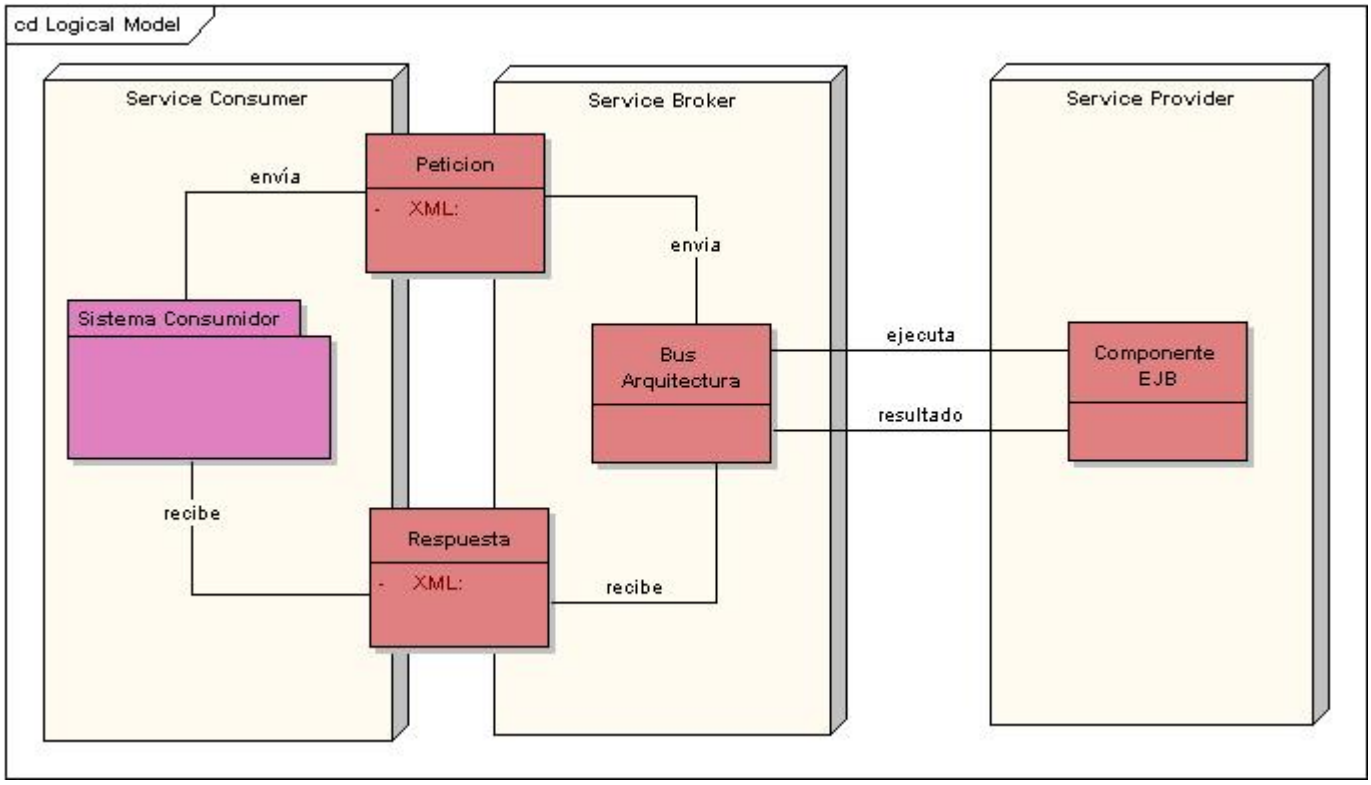

Figura B-12: Diagrama de Clases Nivel 0.

Esta clase ejecuta el servicio, el cual está encapsulado en un componente EJB, de manera de controlar las transacciones a través del mismo servidor. Este componente, donde ya es propiamente tal el servicio, se encuentra en el nodo “Service Provider” y retorna el valor de la operación en forma de mensaje JMS hacia el “Bus Arquitectura”, el cual lo devuelve a la capa del Service Consumer, retornando el valor del mensaje procesado.

En un nivel de detalle más profundo se muestra el diagrama de Clases Nivel 1, el cual explica con más detalle, introduciendo en los componentes que al final serán implementados dentro de la arquitectura. Este diagrama se puede visualizar en la Figura 4-7, el objetivo es visualizar como se comporta un requerimiento a través de las clases que

componen el sistema. Además de se puede apreciar de manera preliminar las clases y paquetes, con los componentes que posee la arquitectura.

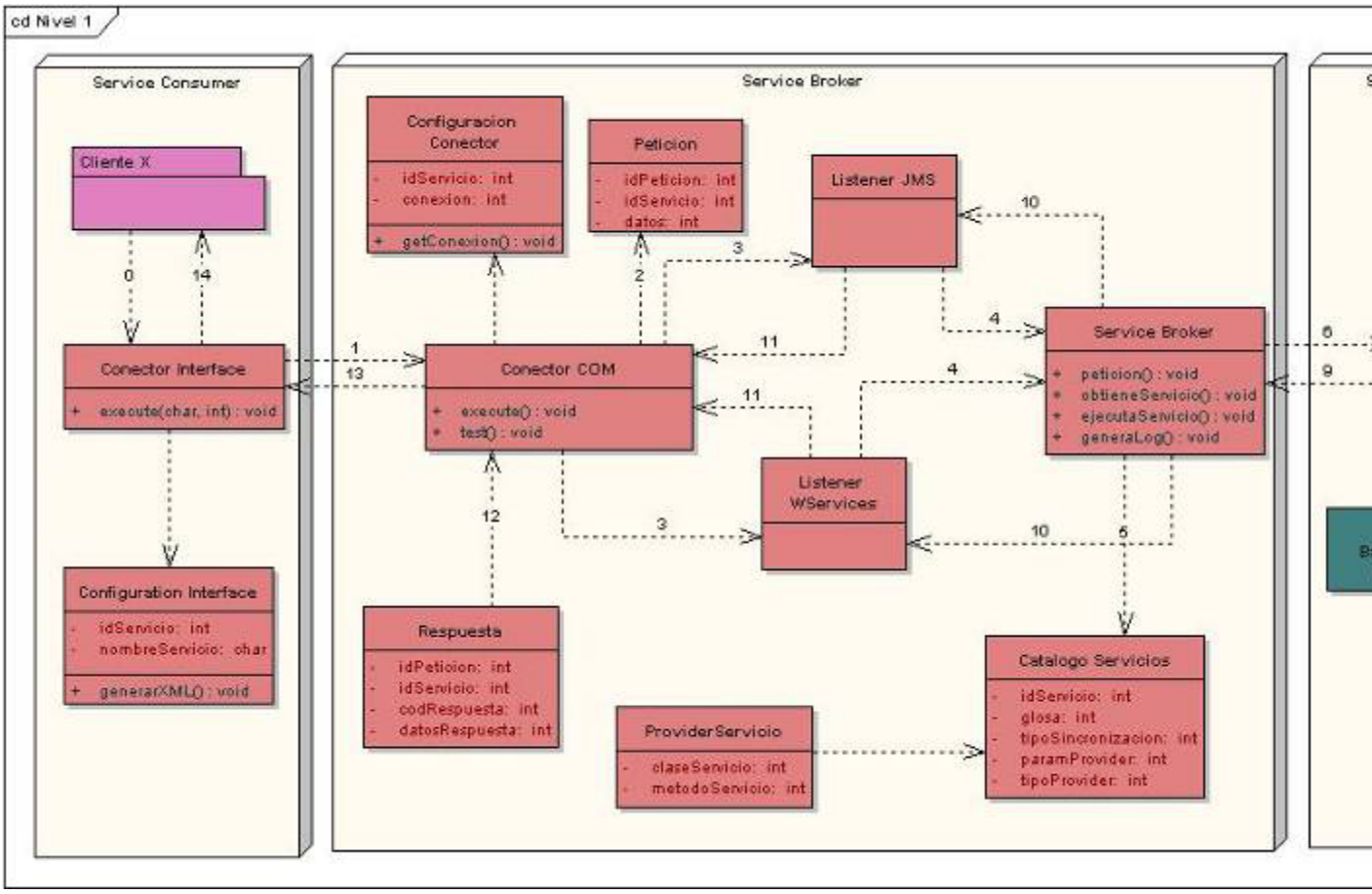

Figura B 13: Diagrama de Clases Nivel 1.

B.1.3 Modelo de Base de Datos.

Para soportar la funcionalidad de seguridad y agregación de servicios, se plantea el siguiente esquema de Bases de Datos.

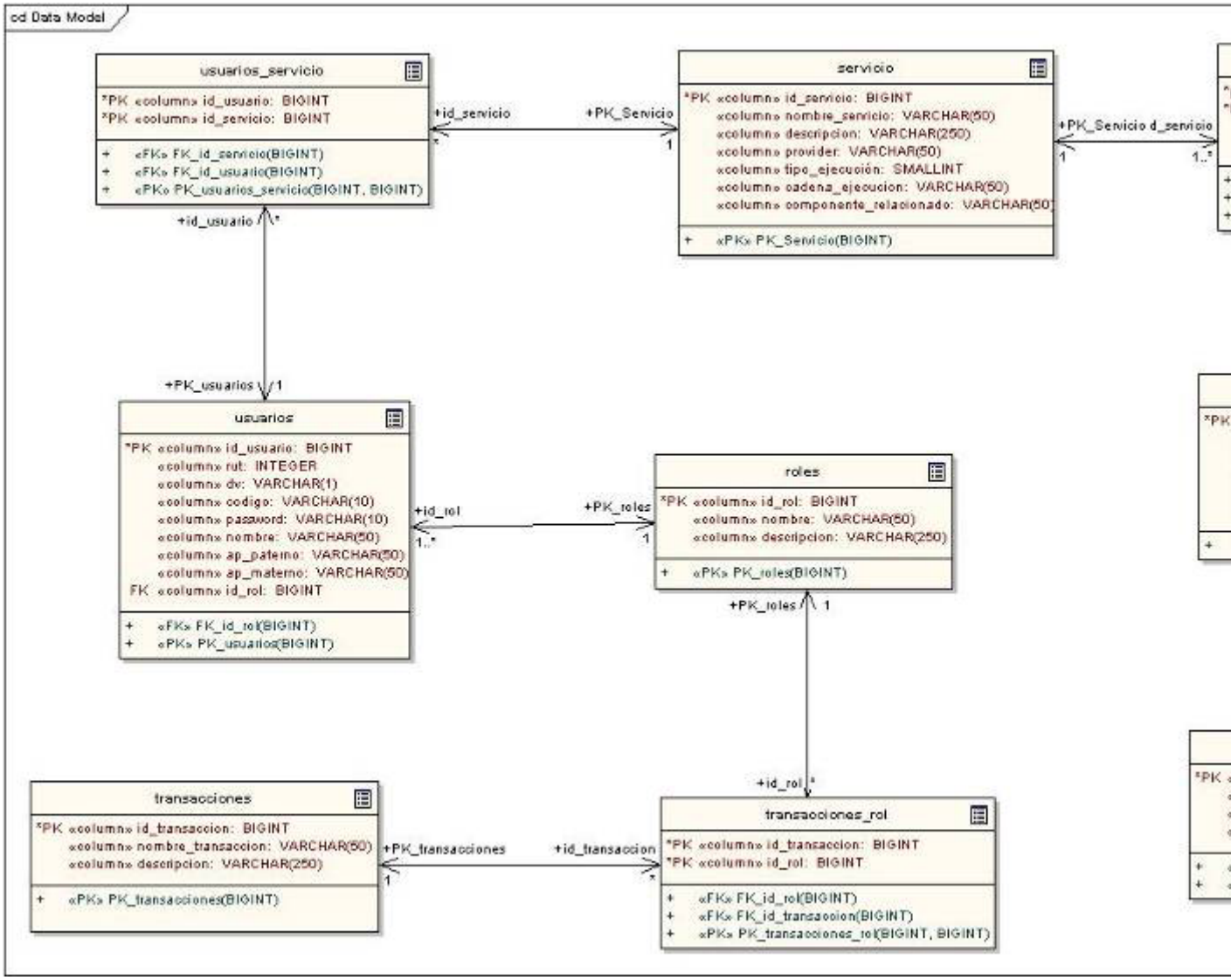

Figura B-14: Modelo de Base de Datos.