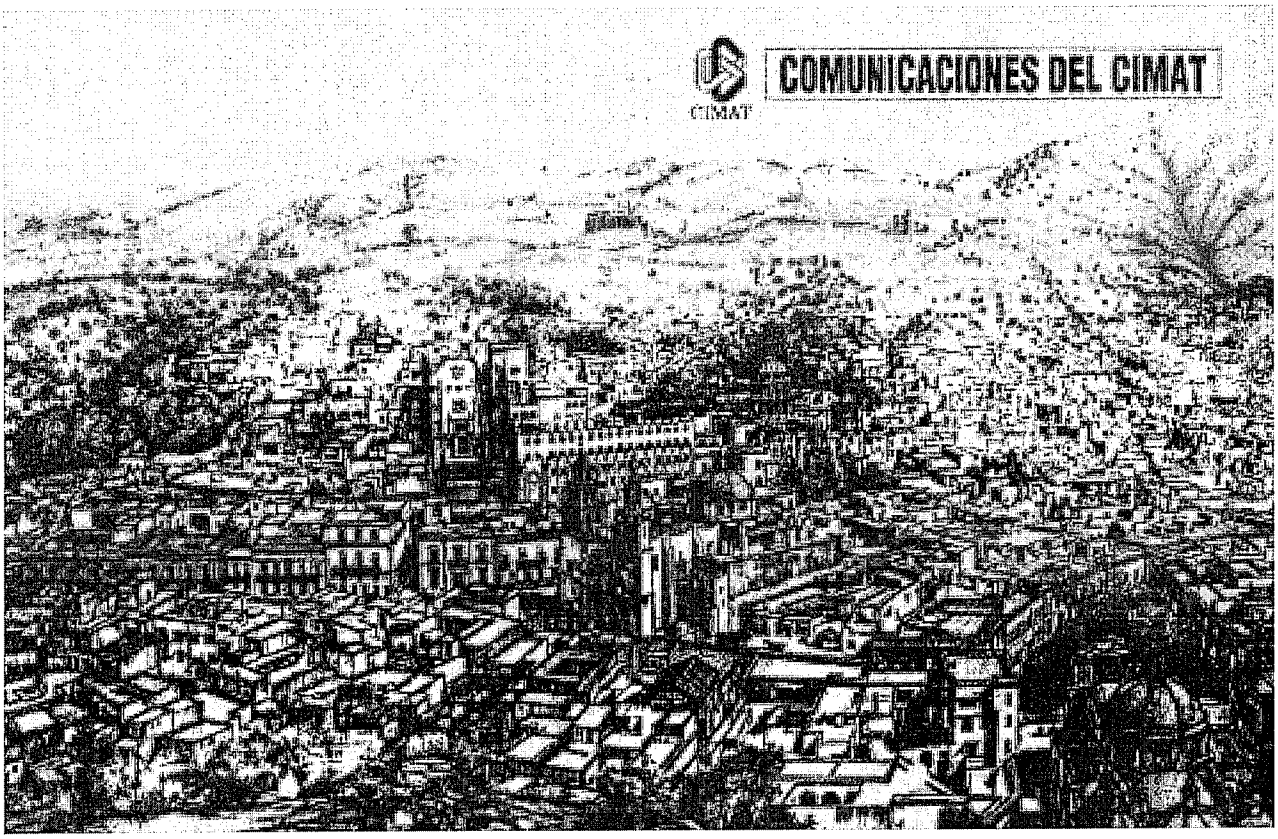
INTEGRACIÓN DE SISTEMAS INFORMATICOS A TRAVÉS DE ARQUITECTURAS ORIENTADAS A SERVICIOS

César Alberto Collí Cuevas



Reporte Técnico

Integración de sistemas informáticos a través de las Arquitecturas Orientadas a Servicios.

César Alberto Collí Cuevas

Versión 1.0
Agosto 2006

Histórico de revisiones

Versión	Fecha	Descripción	Autor
0.1	04.08.06	Primer borrador.	CCC
0.2	10.08.06	Correcciones.	CCC
0.3	20.08.06	Sección 4	CCC
0.4	23.08.06	Conclusiones	CCC
0.5	24.08.06	Correcciones	CCC

Tabla de Contenido

RESUMEN 3
AGRADECIMIENTOS 3
1. Introducción 3
2. Panorama general de SOA 3
2.1. ¿Que es SOA? 3
2.2. Objetivos de SOA 3
2.3. Elementos de SOA 3
2.4. Principios de SOA 3
2.5. Visión interna de los Servicios Web 3
2.6. Contrato de Servicios 3
2.7. Administración de Procesos de Negocio con SOA 3
2.8. Acceso a multicanales con SOA 3
3. SOA y Servicios Web para integración 3
3.1. Introducción a la integración 3
3.2. Integración e interoperabilidad usando Servicios Web 3
3.3. Enfoque de integración basado en servicios. 3
4. Etapas de adopción de SOA 3
4.1. Etapa 1: Experimentando con servicios Web 3
4.2. Etapa 2: Adaptando sistemas existentes para el uso de Servicios Web 3
4.3. Etapa 3: Removiendo dependencias entre sistemas. 3
4.4. Etapa 4: Establecer un Arquitectura Orientada a Servicios Interna. 3
5. Construyendo un servicio Web, paso a paso. 3
5.1.1. Descripción del Problema. 3
5.1.2. Solución Propuesta. 3
5.1.3. Paso 1: Diseñar de los mensajes y tipos de datos. 3
5.1.4. Paso 2: Construir el esquema XML para los tipos de datos 3
5.1.5. Paso 3: Crear la clase de las definiciones de los tipos de datos del servicio Web partir del esquema XML. 3
5.1.6. Paso 4: Crear la Interfase y el Adaptador del servicio Web. 3
5.1.7. Paso 5: Implementar el Servicio Web 3
5.1.8. Paso 6: Ejecutar el Servicio Web. 3
5.1.9. Paso 7: Consumir el Servicio Web. 3
6. Conclusiones 3
7. Glosario 3
8. Referencias 3

RESUMEN

La aparición de los *servicios Web* permite tener una arquitectura común para exponer la funcionalidad de las distintas aplicaciones y sistemas creando nuevos *Procesos de negocio* que permitan crecer a las plataformas tecnológicas, a los departamentos y por su puesto a las compañías.

La aplicación más importante de las arquitecturas orientadas a servicios es la posibilidad de conectar varios sistemas para que automaticen los procesos de negocio de las empresas.

En este documento se presenta las principales características de las arquitecturas orientadas a servicios y cómo estas apoyan la integración de sistemas informáticos para el beneficio de las organizaciones.

AGRADECIMIENTOS

Estado de Guanajuato (CONCYTEG) a través del Proyecto Número GTO-2002-C01-5333 titulado Promoviendo calidad en la industria de software: Recursos humanos, investigación, servicios".

Al Dr. Cuauhtémoc Lemus Olalde por sus valuables recomendaciones dadas para la elaboración de este documento.

1. Introducción

En las últimas décadas los departamentos de sistemas de las empresas han construido una infraestructura que actualmente soporta en gran medida la operación de sus empresas y sus clientes. La gran pregunta fue como desarrollar sistemas para automatizar las operaciones manuales entre sistemas. El resultado de este proceso, ha sido la creación y mantenimiento de un número considerable de aplicaciones al interior de las empresas, cada una responsable de sus propias tareas.

En la era moderna, muchas operaciones de negocio han sido automatizadas, y ahora la gran pregunta es como mejorar la capacidad de estos sistemas para cumplir nuevos requerimientos. Agregando una nueva interfase de usuario, combinando múltiples fuentes de datos en una simple vista, integrando dispositivos móviles, o remplazando las aplicaciones viejas con una mejor, son comunes razones de investigación en nuevos proyectos.

En este contexto los ejecutivos necesitan encontrar maneras para resolver los problemas tales como:

- Costosas e inflexibles tecnologías de integración que causan riesgos inaceptables para la empresa.
- La complejidad incremental de los ambientes de tecnologías de información.
- Aplicaciones de negocio monolíticas que requieren un mantenimiento muy caro.
- La compleja automatización de los procesos de negocio que envuelven a socios y clientes con una inadecuada seguridad.
- Falta de visibilidad y control en procesos de negocios automatizados.

Los negocios exigen crear aplicaciones cada vez más complejas, con menos tiempo y presupuesto que antes. En muchos casos crear estas aplicaciones requiere de funcionalidades ya antes implementadas como parte de otros sistemas. En este punto los arquitectos de software se pueden enfrentar a dos opciones:

- Tratar de reutilizar la funcionalidad ya implementada en otros sistemas. Una labor difícil de realizar, debido a que estos no fueron diseñados para integrarse o sobre plataformas y/o tecnologías incompatibles entre ellas.
- Re-implementar la funcionalidad requerida. Aunque implica más tiempo de desarrollo, es en la mayoría de los casos la más fácil y segura.

La integración de diferentes tecnologías, productos y procesos es un reto que han preocupado en las últimas décadas a los departamentos de sistemas de las organizaciones.

La respuesta al problema de integración de aplicaciones la tienen las Arquitecturas Orientadas a Servicios (*SOA*, por sus siglas en inglés). *SOA* tienen como elemento principal a los *servicios Web* encargados de la comunicación entre las entidades que se quieren integrar.

La aparición de los *servicios Web* [1] permite tener una arquitectura común para exponer la funcionalidad de las distintas aplicaciones y sistemas creando nuevos *Procesos de negocio* que permitan crecer a las plataformas tecnológicas, a los departamentos y por su puesto a las compañías.

2. Panorama general de SOA.

2.1. ¿Que es SOA?

La Arquitectura Orientada a Servicios (SOA, por sus siglas en ingles) es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

SOA proporciona una metodología y un marco de trabajo para documentar las capacidades negocio y puede dar soporte a las actividades de integración y consolidación. [2]

Antecedentes de SOA

En las últimas décadas la arquitectura de las aplicaciones en tecnología de información ha evolucionado para satisfacer las necesidades de negocio. En la siguiente figura se muestra la evolución de estas arquitecturas.

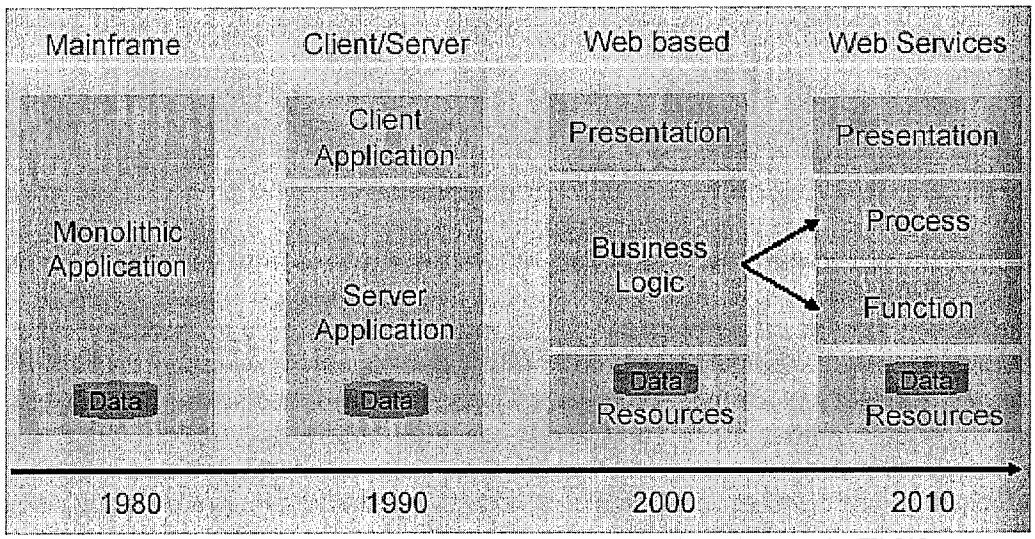


Figura 1. Evolución de arquitecturas de aplicaciones de TI [3]

Las viejas aplicaciones monolíticas se caracterizaban por no ser estructuradas e inflexibles, diseñadas con leguajes de programación de tercera o cuarta generación, corriendo en sistemas mainframe, estas aplicaciones fueron reemplazadas en los 90 por los sistemas cliente-servidor.

10 años después otro paradigma llego a establecerse como resultado de desarrollos tecnológicos significativos y emergiendo de los requerimientos relacionados al negocio: Los negocios basados en la Web. La evolución de la tecnología de Internet seguramente ha llegado a ser el principal conductor de las nuevas arquitecturas de tecnologías de información de hoy en día impulsando las arquitecturas multicapas. Las arquitecturas multicapa comprende la capa de presentación que provee principalmente clientes Web o otros tipos de clientes dedicados a presentar la información; la capa de negocio, la cual representa la lógica de negocio de la aplicación donde se

ejecutan los procedimientos que realizan las reglas de negocio; y la capa de datos que contiene los procedimientos de acceso a las bases de datos.

La arquitectura orientada a servicios ahora permite un nuevo paradigma para el diseño y operación de aplicaciones de negocio. La separación en funciones y procesos. En este paradigma los procesos de negocios y actividades pueden estar soportados eficientemente por la composición de servicios Web coordinados, los cuales representan funcionalidades de negocio.

Sin embargo este paradigma, el cual puede ser descrito como un modelo de programación de dos niveles tendrán un impacto significativo en el diseño ya que en esta fase la composición de servicios Web significan nuevos retos que conciernen a la coordinación de estos servicios y el manejo coordinado de transacciones.

En una aplicación construida con SOA, todas las funciones están definidas como servicios independientes con interfases invocables bien definidas. Pueden ser llamados en secuencias definidas para conformar un proceso de negocio.

Todos los servicios son independientes, operan como cajas negras. Los componentes externos no conocen ni les interesa como desempeñan su función, solo que entreguen el resultado esperado.

Los servicios pueden ser funciones de bajo nivel o complejos de alto nivel (granularidad fina o gruesa), y basados en esta definición existen diferencias muy reales en desempeño, facilidad de mantenimiento y reutilización. El nivel de granularidad es una expresión de la riqueza funcional del servicio. Por ejemplo, entre mayor granulación, más riqueza funcional ofrecida. Los servicios típicamente son funciones de negocio de grano grueso.

El proceso de definir servicios es normalmente una conversación XML realizada por objetos dentro de un alcance mayor, el de la estructura de la aplicación. Este es realmente el trabajo a realizar, el desarrollo de una estructura basada en componentes, en la cual los servicios son definidos como un conjunto de componentes reutilizables los cuales pueden usarse para construir nuevas aplicaciones o integrar programas existentes.

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)* para implementar el mecanismo de comunicación.
- Se podia utilizar *CORBA (Common Object-Request Broker Architecture)*.
- Se podía utilizar un protocolo definido de comunicación particular entre dos aplicaciones.

Comparado con las técnicas anteriores los servicios Web son mucho más fáciles de aprender y usar.

El objetivo de una arquitectura SOA es proveer de la transparencia en la localización del servicio 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. La aplicación más importante de SOA es la posibilidad de conectar varios sistemas para que automaticen los procesos de negocio de las empresas.

2.2. Objetivos de SOA

Exponer procesos de negocio como servicios es la clave a la flexibilidad de la arquitectura. Esto permite que otras piezas de funcionalidad (incluso también implementadas como servicios) hagan uso de otros servicios de manera natural, sin importar su ubicación física.

Así un sistema evoluciona de una manera independiente. La Arquitectura Orientada a Servicios (SOA) resultante, define los servicios de los cuales estará compuesto el sistema, sus interacciones, y con que tecnologías serán implementados. Las interfaces que utiliza cada servicio para exponer su funcionalidad son gobernadas por contratos, que definen claramente el conjunto de mensajes soportados, su contenido y las políticas aplicables. [4]

2.2.1. Desde el punto de vista empresarial.

Cuando una empresa decide hacer uso de una arquitectura SOA es porque tiene objetivos específicos de negocio que cubrir, reducir costos, aumentar ingresos, mejorar la productividad, comunicación inter-empresarial con varias empresas y ajustar los sistemas a los requerimientos de 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 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 y departamentos, así como también reducir el retorno de inversión, es decir, conseguir una mayor rentabilidad de las inversiones tecnológicas.

Beneficios para el negocio:

- **Eficiencia.** Transforma los procesos de negocio en servicios compartidos con un menor costo de mantenimiento.
- **Capacidad de respuesta.** Rápida adaptación y despliegue de servicios, claves para las demandas de los clientes, accionistas y empleados.
- **Adaptabilidad.** Facilita la adopción de cambios añadiendo flexibilidad y reduciendo el esfuerzo.

2.2.2. Desde el 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 procesos de negocio.

Las aplicaciones basadas en SOA utilizan tecnología totalmente estándar como es XML y servicios Web para la mensajería. Estándares como *SOAP*, *Web Services Description Language (WSDL) y Business Process Execution Language (BPEL),* estandarizan así la compartición de información, el modelo de integración de procesos y la cooperación entre aplicaciones [5].

Realizando aplicaciones orientadas a servicios 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.

Como una arquitectura SOA se basa en estándares, el tiempo de aprendizaje de utilización de las tecnologías sobre las que se apoya se reduce drásticamente.

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 costo de mantenimiento e integración.
- **Beneficios en el desarrollo** ya que las aplicaciones son reutilizables, mas fáciles de mantener y tienen la capacidad de ampliación de las funcionalidades del sistema, exponiéndolas de una forma segura.

2.3. Elementos de SOA

Esta arquitectura presenta una forma de construir sistemas distribuidos que entreguen a la aplicación funcionalidad como servicios para aplicaciones de uso final u otros servicios.

En la figura 2 se muestra el stack de la arquitectura y los elementos que podrían observarse en una arquitectura orientada a servicios.

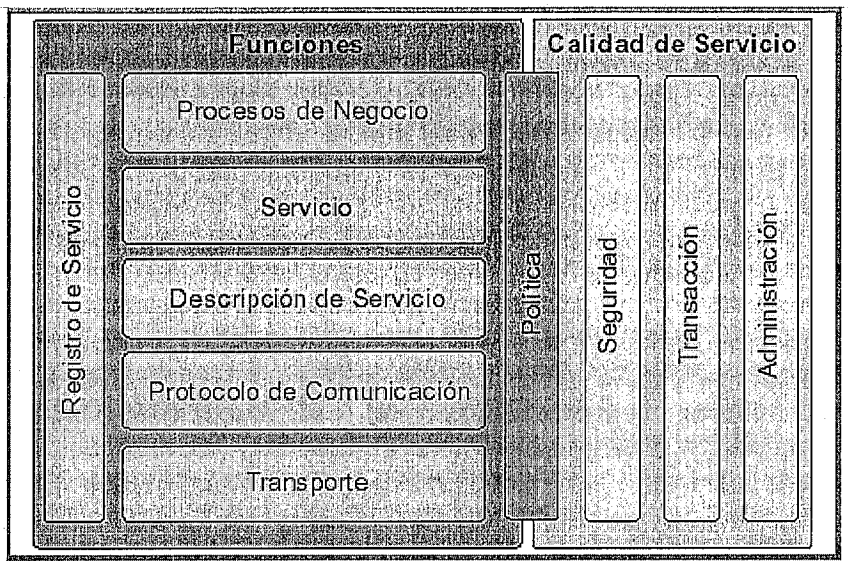


Figura 2. Elementos de una arquitectura orientada a servicios (SOA) [6]

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

- Funciones
  - **Transporte:** es el mecanismo utilizado para llevar las demandas de servicio desde un consumidor de servicio hacia un proveedor de servicio, y las respuestas desde el proveedor hacia el consumidor.
  - **Protocolo de comunicación de servicios:** es un mecanismo acordado a través del cual un proveedor de servicios y un consumidor de servicios comunican qué está siendo solicitando y qué está siendo respondido.
  - **Descripción de servicio:** es un esquema acordado para describir qué es el servicio, cómo debe invocarse, y qué datos requiere el servicio para invocarse con éxito.
  - **Servicios:** describe un servicio actual que está disponible para utilizar. Pueden participar cualquier número de servicios. Cada servicio tiene una funcionalidad a la que pueden acceder el resto de servicios y clientes.
  - **Procesos de Negocio:** es una colección de servicios, invocados en una secuencia particular con un conjunto de reglas, para satisfacer un requerimiento de negocio.
  - **Registro de Servicios:** es un repositorio de descripciones de servicios y datos que pueden utilizar proveedores de servicios para publicar sus servicios, así como consumidores de servicios para descubrir o hallar servicios disponibles.
- Calidad de Servicio
  - **Política:** es un conjunto de condiciones o reglas bajo las cuales un proveedor de servicio hace el servicio disponible para consumidores.

- **Seguridad:** en un conjunto de reglas que pueden aplicarse para la identificación, autorización y control de acceso a consumidores de servicios.
- **Transacciones:** es el conjunto de atributos que podrían aplicarse a un grupo de servicios para entregar un resultado consistente.
- **Administración:** es el conjunto de atributos que podrían aplicarse para manejar los servicios proporcionados o consumidos.

Las colaboraciones en SOA siguen el paradigma *find, bind and invoke*, donde un consumidor de servicios realiza la localización dinámica de un servicio consultando el registro de servicios para hallar uno que cumpla con un determinado criterio. Si el servicio existe el registro proporciona al consumidor la interfaz de contrato y la dirección del servicio proveedor. El siguiente diagrama ilustra las entidades (roles, operaciones y artefactos) en una arquitectura orientada a servicios donde estás colaboran.

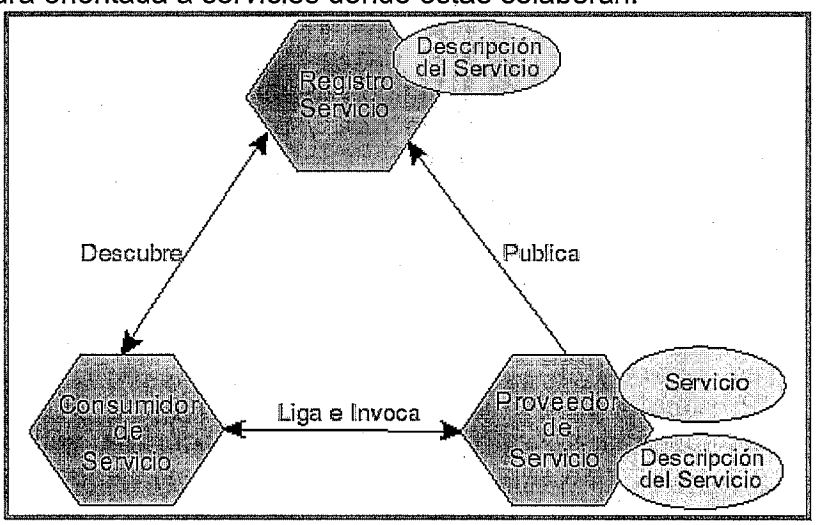


Figura 3. Colaboraciones en SOA:

Cada entidad puede tomar el rol de consumidor, proveedor y/o registro:

- **Un consumidor de servicios** es una aplicación, un módulo de software u otro servicio que requiere un servicio, y ejecuta el servicio de acuerdo a un contrato de interfaz.
- **Un proveedor de servicios** es una entidad direccionable a través de la red que acepta y ejecuta consultas de consumidores, y publica sus servicios y su contrato de interfaces en el registro para que el consumidor de servicios pueda descubrir y acceder al servicio.
- **Un registro de servicios** es el encargado de hacer posible el descubrimiento de servicios conteniendo un repositorio de servicios disponibles y permitiendo visualizar las interfaces de los proveedores de servicios a los consumidores interesados.

Las operaciones son:

- **Publicar.** Para poder acceder a un servicio se debe publicar su descripción para que un consumidor pueda descubrirlo e invocarlo.

- **Descubrir.** Un consumidor de servicios localiza un servicio que cumpla con un cierto criterio consultando el registro de servicios.
- **Ligar e invocar.** Una vez obtenida la descripción de un servicio por parte de un consumidor, éste lo invoca de acuerdo a la información en la descripción del servicio.

Finalmente, los artefactos en una arquitectura orientada a servicios son:

- **Servicio.** Un servicio que está disponible para el uso a través de una interfaz publicada y que permite ser invocado por un consumidor de servicios.
- **Descripción de servicio.** Una descripción de servicio especifica la forma en que un consumidor de servicio interactuará con el proveedor de servicio, especificando el formato de consultas y respuestas desde el servicio. Esta descripción también puede especificar el conjunto de precondiciones, poscondiciones y/o niveles de calidad de servicio.

2.4. Principios de SOA

Los principios de SOA son algo que lo arquitectos y desarrolladores deben de tener en cuenta al implementar este tipo de arquitectura. Estos principios son [7]:

- **Principio 1:** Los servicios son el concepto central de las Arquitecturas Orientadas a Servicios.
- **Principio 2:** Cada servicio esta definido por un contrato formal que separa claramente la funcionalidad que esta siendo provista de la implementación técnica.
- **Principio 3:** Los servicios deberán solamente interactuar con otros servicios a través de sus interfaces públicas bien definidas.
- **Principio 4:** Los servicios deberán ser accesibles a través de tecnologías estandarizadas que están disponibles a través de una amplia gama de ambientes. Esto también implica que los mecanismos de invasión (protocolos, transporte, descripciones de servicio, etc.) deberán cumplir los estándares especificados.
- **Principio 5:** Los servicios deberán estar definidos en un nivel de abstracción que corresponda con las actividades del mundo real y las funciones de negocios.
- **Principio 6:** Los servicios deberán ser significativos para el consumidor del servicio.
- **Principio 7:** Los servicios deben estar débilmente acoplados.
- **Principio 8:** Las colecciones de servicios relacionados deberán usar el mismos tipos del documento XML para facilitar el intercambio de información entre los servicios relacionados, y la estructura y semántica debe ser aceptado y bien entendido.
- **Principio 9:** Los servicios deberán desempeñar tareas discretas y proveer interfaces simples para acceder a la funcionalidad.
- **Principio 10:** Los servicios deberán proveer los metadatos que definen sus capacidades y restricciones, y estos metadatos deberán estar disponibles en el repositorio, el cual puede ser accedido a través de los

servicios. Este repositorio debe mostrar las definiciones de los servicios para ser inspeccionados sin necesidad de acceder a los servicios. Esto debe permitir a los consumidores de servicios localizar dinámicamente e invocar el servicio.

Para implementar un Arquitectura Orientada a Servicios de forma pura los arquitectos y desarrolladores deberán respetar los principios que rigen la misma arquitectura.

Además de los principios expuestos, también es necesario considerar lo siguiente al construir sistemas orientados a servicios:

- **Sin estado**. Los servicios deberían apuntar a un intercambio de mensajes que demuestren toda la información necesaria para recibir servicios para restablecer el contexto de una interacción en una conversación multimensaje. Los servicios sin estado son fáciles de escalar y simplifican mucho la tolerancia de fallas mediante redundancia.
- **Mensajes enriquecidos**. Los costos de la comunicación generalmente son elevados. Por los tanto, debemos apuntar a los protocolos que impliquen mensajes enriquecidos que den como resultado interacciones detalladas, disminuyendo de forma eficaz el número de veces que un servicio tiene que cruzar toda la red.
- **Administración de estados**. En cuanto al tradicional diseño de aplicación, la implementación de servicio deberá delegar todos los aspectos de la administración de estados en almacenes de datos.
- **Envió de mensajes**. No deberá haber suposiciones sobre los mecanismos de envío empleados por los servicios. Como resultado, ninguna información específica de envío se deberá filtrar de las implementaciones de servicio, cruzar los límites y mostrarse a través de contenidos del mensaje.
- **Acoplamiento de roles**. Los arquitectos deberán tener siempre en mente que en una arquitectura orientada a servicios no existen "consumidor", "proveedor", "cliente", "servidor" y demás. Estos son roles que existen a nivel de aplicación no de bloques de construcción de la arquitectura. En SOA, solo existen los servicios que intercambian mensajes. Considerar a un par de servicios como cliente-servidor presenta una forma de acoplamiento que podría en definitiva ser difícil de romper.

Los servicios son una abstracción sensata para encapsular y administrar el creciente nivel de complejidad de las aplicaciones distribuidas. Lo bueno de la orientación a servicios es que las pautas y principios de la arquitectura son sistemáticos en su totalidad desde un proceso de sistema operativo hasta un servicio que encapsula un proceso empresarial completo o aún toda una organización.

2.5. Visión interna de los Servicios Web

Según la "World Wide Web Consortium (W3C)" la definición de servicio Web es:

"Es un sistema de software identificado por medio de una URL, que publica interfaces que son definidas usando XML. Esta definición de interfaz puede ser consumida por otros sistemas de software. Estos sistemas pueden interactuar con el servicio Web de la manera prescrita en la definición, usando XML, utilizando mensajes cubiertos por los protocolos de Internet."

Generalmente, los servicios proporcionan tanto la lógica de negocio como el manejo de estado (datos), relevantes a la solución del problema para el cual fueron diseñados. La manipulación del estado es gobernada por las reglas de negocio.

Un servicio funciona como una aplicación independiente, teniendo sus propias reglas de negocio, datos, procedimientos de administración y operación. Expone toda su funcionalidad utilizando una interfaz basada en mensajes.

La comunicación hacia y desde el servicio, es realizada utilizando mensajes y no llamadas a métodos. Estos mensajes deben contener o referencias toda la información necesaria para ser entendidos.

Al contrario de las arquitecturas orientadas a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables.

Un servicio es la evolución en complejidad de un componente distribuido, y se diferencian en:

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

En cuanto a los servicios Web es importante tener en cuenta que no son los responsables del comportamiento de un servicio, sino que se centran en como se puede acceder a dicho comportamiento que ofrece un servicio.

Los servicios otorgan funcionalidad comercial a sus consumidores. Los analistas y los usuarios comerciales prefieren pensar en términos de las entidades que utilizan. Entienden las relaciones entre estas entidades y las acciones que se realizan sobre estas.

La mayoría de los servicios resumen entidades. Por más concretas que sean estas entidades para los diseñadores del servicio, para el consumidor son abstractas. El consumidor sin embargo entiende lo que representan: Cliente, Orden, Informe del Problema, etc. La entidad esta hecha de la combinación de todo el estado, rendimiento y relaciones con otras entidades.

El servicio muestra vistas sobre las entidades. El consumidor puede obtener la vista de una dirección, o una vista financiera, o una vista de historial sobre el cliente. Estas vistas son instantáneas de la entidad, tomadas desde perspectivas específicas. Estas vistas son concretas. Tienen un esquema bien definido exponiendo atributos o propiedades de la entidad. Debe notarse sin embargo que las vistas no necesariamente presentan un subconjunto de información sobre entidad. Pueden contener información de entidades relacionadas y superponerse a las vistas sobre esas entidades. La vista sobre una orden puede contener el nombre y dirección del cliente y también las descripciones del producto para utilizarse en la interfaz del usuario o para llenar un documento.

El servicio también tiene métodos o acciones. Muchas de estas están relacionadas con una entidad. Por ejemplo, "liberar orden" o "actualizar cliente" se refieren claramente a entidades específicas. Y si estas acciones requieren información de entrada, ésta con frecuencia proviene de vistas sobre otras entidades. Por ejemplo el pedido a un proveedor de crear una orden de venta se realiza utilizando la información de la propuesta de compra. Una vista sobre la propuesta de compra se convierte en un pedido de orden de venta.

Las entidades, vistas y acciones son artefactos que podemos describir, que el usuario puede entender y que pueden ayudarnos a describir servicios de manera que entiendan los analistas comerciales.

Muchas entidades pueden identificarse utilizando atributos clave, sin embargo, con frecuencia pueden ser identificas solo usando otros atributos. Los clientes pueden identificarse usando su número de documento de identidad. Se puede obtener una lista de clientes por medio de un código postal, la región, o el gerente de cuenta. Podemos formalizar esto a través de la referencia. Una referencia define una instancia única o una lista de instancias. Una referencia, como la vista, tiene un esquema.

Las acciones expuestas por el servicio aceptan referencias para identificar a las entidades sobre las que operan. Las referencias son una manera más formal de definir la posible entrada y una manera más original también.

Las referencias formulan preguntas a los servicios. Esta pregunta puede ser tan simple como la lista de atributos clave o tan complejos como la formula de las condiciones. Con frecuencia una referencia puede consistir en la combinación de atributos obligatorios y opciones que permite a los clientes que tienen diferente información obtener las mismas vistas y usar las mismas acciones a través de las mismas interfases. El diseño de los esquemas de las

referencias es tan importante y complejo como el diseño de los esquemas de vistas.

Si se tiene la vista de una entidad no debe ser difícil construir una referencia a una entidad relacionada, por ejemplo, el cliente de esa orden. En otras palabras, se puede transformar la vista de la orden en una referencia al cliente que se refiere a la entidad del cliente. Esto efectivamente nos da una relación entre la vista de orden y la entidad del cliente.

La relación usa datos en una vista para construir una referencia a otra entidad. Esta relación puede definirse usando los esquemas de una vista y una referencia. Por lo tanto las relaciones no están limitadas a un solo servicio. Pueden definir relaciones entre entidades resumidas en servicios diferentes.

Si podemos describir estas entidades, las vistas sobre estas entidades, las relaciones entre estas entidades y las acciones sobre estas entidades en los términos entendibles para gente de negocios y software, tendremos una descripción adecuada. Esta descripción no sólo describe los artefactos y relación entre ellos, sino también el funcionamiento, lo que las personas y el software puede hacer con ellos.

Las principales consideraciones principales que deben tomarse en el diseño, implementación y administración de servicios son las siguientes:

- **Los servicios deben estar débilmente acoplados.** Las dependencias entre el proveedor del servicio y el consumidor del servicio deben ser mínimas. Idealmente, el consumidor del servicio debe ser capaz de usar el servicio tan solo entendiendo el contrato especificado por el proveedor servicio y bajo ninguna circunstancia el consumidor del servicio debería requerir información de la implementación interna del servicio. Los servicios además no deben restringir al consumidor de este usar una tecnología específica, de un producto o plataforma de desarrollo.
- **Los contratos de los servicios deben estar bien definidos.** Cada servicio debería especificar claramente las capacidades del servicio y como invocar los servicios de una manera ínter operable, y claramente separar las interfaces de la implementación interna.
- **Los servicios deben ser significativos para los consumidores.** Debe estar en un nivel apropiado de abstracción y deberá capturar la esencia del servicio de negocio. Además deberá tener un lenguaje orientado al negocio y evitar detalles técnicos tales como estructuras internas o convenciones.
- **Los servicios deben estar basados en estándares**.

2.6. Contrato de Servicios

Cada servicio tiene una interfase formal bien definida llamada contrato de servicio que claramente define lo que lo servicios hacen y separa claramente las interfaces externas de la implementación técnica interna.

Los elementos del contrato de servicio son [8]:

- **Nombre del servicio.** Debe ser un nombre amigable.
- **Versión.** Soporta el ciclo de vida del producto.
- **Precondiciones.** Condiciones que deben ser satisfechas antes de usar el servicio.
- **Clasificación.** Notas y palabras claves que identifican el dominio del negocio en el que servicio aplica.

Además, para cada operación del servicio deberán especificarse los siguientes elementos:

- **Nombre de la operación.** Debe ser un nombre amigable.
- **Precondiciones.** Condiciones que deben satisfacerse antes de invocar la operación.
- **Postcondiciones.** Cambios de estado que ocurren cuando la operación se completa exitosamente.
- **Perfil de entrada de datos.** Nombre, tipo, estructura, restricciones, y significado de cada documento de entrada.
- **Perfil de salida de datos.** Nombre, tipo, estructura, restricciones y significado para cada documento de salida.
- **Perfil de interacción.** Determina que modos de invocación realiza la operación, es decir, si es un tipo de invocación de solicitud-respuesta, de una sola vía, asíncrona, etc.
- **Condiciones de excepción y manejo de errores.** Códigos de los errores correspondientes a los mensajes de error. Cambios de estado que ocurren bajo condiciones de error.
- **Perfil de seguridad.** Políticas de seguridad que aplican a su operación incluyendo técnicas de autenticación y autorización, lista de control de accesos, etc.
- **Perfil transaccional y semántica de recuperación.** Políticas de transacciones, si aplica.
- **Nivel de aceptación.** Por ejemplo, garantías de disponibilidad, tiempos de respuesta, etc.

La importancia de tener los contratos de servicios bien definidos es un principio vital para las arquitecturas orientadas a servicios ya que son el punto de partida para la lograr la integración.

Aunque mucha de esta información puede suministrarla el diseñador de la interfaz de servicio similar a cómo se provee el WSDL actualmente, una extensión del WSDL no puede solucionar todas las necesidades. Las relaciones entre entidades ofrecidas por diferentes servicios no deben almacenarse en ninguna, pero requerirían almacenamiento externo a estos servicios. La información específica, como la disponibilidad actual de las relaciones y acciones, como la información creada en tiempo de ejecución, como tareas en trabajos de colaboración, requieren interrogatorio del tiempo de ejecución y por ende descripción del tiempo de ejecución. Esto esta relacionado con los contratos de servicios y la infraestructura requerida para proveer información del tiempo de ejecución sobre el estado de una conversación en particular.

Ya que las vistas son lo que el diseñador expone sobre las entidades, mucha información esta disponible en los sistemas actualmente. Cada servicio que fue diseñado sobre las entidades suministra la mayoría de la información que se necesita y en muchos otros sistemas el análisis ha sido hecho y capturado en alguna clase de metadatos.

Los valores de estas referencias provienen de la vista sobre la orden. Los nombres de los elementos y atributos pueden ser diferentes a veces entre la vista y la referencia. A veces es útil combinar valores que no provienen del original, si desea restringir el resultado a una clasificación determinada o si necesita una combinación de vistas como la fuente.

Es importante proveer un modelo que, como WSDL, puedan usar herramientas y consumidores de servicios. Sin embargo, es igualmente importante no sólo proveer información fija. El esfuerzo necesita alinearse con este pensamiento sobre contratos de servicio y debe incluir definiciones dinámicas y redefiniciones de vistas, acciones, referencias, relaciones e incluso entidades. Como proveer disponibilidad actual de acciones y relaciones sobre un caso particular o proveer información sobre un caso de flujo imprevisto.

2.7. Administración de Procesos de Negocio con SOA

El concepto de administración de procesos de negocio (o Business Process Management en ingles) engloba todas las actividades que son parte del ciclo de vida de un proceso de negocio, tales como el descubrimiento, diseño, simulación, deployment, ejecución, interacción, monitoreo, control, análisis y optimización de procesos de negocios[9].

Cada vez más, las necesidades de las empresas se están orientando hacia procesos más complejos que engloban diferentes departamentos, filiales o socios, que pueden incluso estar geográficamente distribuidos. Cada una de estas *entidades* posee sus propios procesos que pueden ser más o menos heterogéneos y complejos. Para representar este tipo de procesos se necesitan sistemas más potentes que los sistemas tradicionales, los que se conocen como Business Process Management Systems (BPMS).

Es así que del modelado de procesos y de la arquitectura orientada a servicios, más precisamente del uso de servicios Web surge un nuevo concepto denominado composición de servicios. Esta composición no solo permite modelar el proceso de negocio sino que también maximiza las potencialidades que SOA brinda a través de la integración de datos y aplicaciones.

La composición de servicios implica encontrar un mecanismo que permita a dos o más de ellos cooperar entre si para resolver requerimientos que van más allá del alcance de sus capacidades individuales. Algunos de los requisitos básicos que se deben cumplir son:

- Interacciones asíncronas con servicios: de manera de dar la posibilidad de crear procesos que transcurran durante largos períodos de tiempo.
- Interacciones simultaneas entre servicios: esto introduce el procesamiento en paralelo lo cual redunda en un aumento considerable de la eficiencia.
- Manejo de excepciones: un proceso basado en múltiples servicios puede fallar en su ejecución si al menos uno de sus componentes falla. Se debe tener en cuenta la ocurrencia de errores y proveer mecanismos para manejarlos.
- Integridad transaccional: un proceso de larga duración puede eventualmente fallar, en este caso puede requerirse que parte de las acciones ya efectuadas se deshagan.

Dos términos comúnmente usados para referirse a la colaboración entre servicios son orquestación y coreografía. Ambos están directamente relacionados con la composición pero se enfocan en aspectos complementarios de la interacción entre servicios. Si bien para los fines de este proyecto el concepto relevante es el de orquestación, vale la pena hacer mención a ambos para de esa forma visualizar sus similitudes y diferencias.

Orquestación

Un proceso se puede considerar una orquestación de servicios cuando es controlado totalmente por una única entidad. Este proceso define completamente las interacciones con los servicios componentes y la lógica requerida para conducir correctamente esas interacciones. Este tipo de proceso puede entenderse como privado y ejecutable ya que solo la entidad que está orquestando el proceso conoce el flujo de control e información que sigue el proceso que se está orquestando. De esta manera se crea un proceso que utiliza diferentes servicios manipulando la información que fluye entre ellos, convirtiendo, por ejemplo los datos de salida de algunos servicios en datos de entrada de otro. Aquí, cada entidad que participa implementa y controla su propio proceso [9].

Coreografía

Un proceso es una coreografía de servicios cuando define las colaboraciones entre cualquier tipo de aplicaciones componentes, independientemente del lenguaje o plataforma en el que estén definidas las mismas. Un proceso de coreografía no es controlado por uno solo de los participantes. A diferencia de la orquestación, la coreografía puede verse como un proceso público y no ejecutable. Público porque define un comportamiento común que todas las entidades participantes deben conocer, y no ejecutable porque está pensando para verse más bien como un protocolo de negocio que dicta las reglas para que dichas entidades puedan interactuar entre sí [9].

2.8. Acceso a multicanales con SOA

El principal propósito de muchas organizaciones es el de proporcionar los servicios a sus clientes. A menudo estas organizaciones usan distintos canales

para hacer llegar los servicios hasta el alcance de sus clientes para asegurar la satisfacción de estos. Por ejemplo un banco ofrece sus servicios de gran variedad de maneras según la conveniencia de sus clientes ya que estos pueden usar la Web, los cajeros automáticos, o la ventanilla del banco para realizar sus operaciones bancarias.

La figura 4 ilustra un ejemplo de multicanales de acceso en los cuales los servicios de las organizaciones pueden ser expuestos para cumplir un objetivo específico.

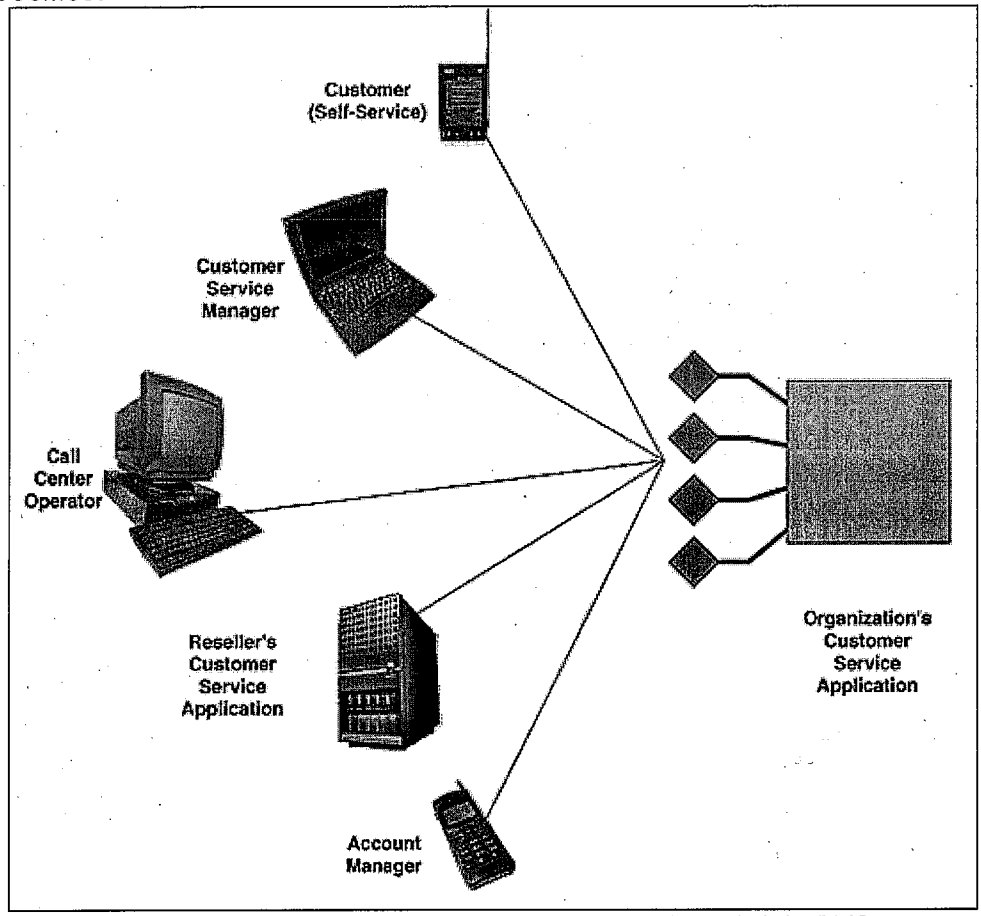


Figura 4. Acceso a multicanales con servicios Web. [10]

Los servicios Web ofrecen una buena manera para el acceso a multicanales, por que estos pueden ser accesibles desde varios tipos de clientes de software incluyendo la Web, dispositivos móviles

3. SOA y Servicios Web para integración.

3.1. Introducción a la integración

Antes de que podamos apreciar completamente como y porque los servicios Web facilitan la integración, necesitamos revisar algunas causas de negocio y factores técnicos que hacen indispensable la integración. En general, concluye en el hecho de que las técnicas de integración o productos tienen un trabajo de reconciliación de las diferencias entre los múltiples sistemas de información cuando quiera que estos sistemas necesiten interactúa. Por su puesto, si todas estas diferencias se dan en una sola dimensión, como por ejemplo en la reconciliación de datos entre sistemas, entonces el problema no deberá ser difícil de resolver. Sin embargo, la integración requiere resolver múltiples capas de negocios y requerimientos técnicos entre sistemas, y que fueron desarrollados por diferentes equipos de desarrollo usando diferentes tecnologías y que resuelven diferentes problemas del negocio, entonces es cuando la complejidad de la integración aumenta.

Algunas causas por las que las organizaciones necesitan invertir en SOA con servicios Web para lograr la integración son [11]:

- **Fusiones y adquisiciones.** Esta actividad resulta en múltiples sistemas de información que manejan similares transacciones y que necesitan ser consolidadas antes de que el valor de negocio de estas fusiones puedan ser completamente realizadas.
- **Reorganización Interna.**
- **Consolidación de aplicaciones.** En este escenario, múltiples sistemas de información manejan transacciones similares, entonces estos necesitan ser consolidados o remplazados para ahorrar dinero, reducir costos, y modernizar las operaciones de negocio. Por ejemplo considérese una compañía de telecomunicaciones que tiene una docena de diferentes sistemas de facturación.
- **Datos inconsistentes, duplicados o fragmentados.** Algunas veces los datos importantes del negocio son propagados en muchos sistemas y deben estar consolidados y limpiados para facilitar la toma de decisiones.
- **Nuevas estrategias de negocios.** Las compañías innovadoras frecuentemente implementan nuevas estrategias de negocios que redefinen el ambiente de negocio pero también requieren de sistemas de información para trabajar de maneras novedosas. Eventualmente, otras compañías en la misma industria tienen que adoptar los mismos cambios para permanecer en la competencia. Por ejemplo las relaciones entre los bancos.
- **Cumplir con nuevas regulaciones de gobierno.** Las nuevas regulaciones del gobierno requieren redefinir los procesos de negocio para proteger a los clientes y cumplir los nuevos requerimientos. Ejemplo, las tasas de interés y nuevos impuestos creados.
- **Eficientizar los procesos de negocio.** Los viejos procesos de negocio donde los datos requeridos son manualmente capturados en múltiples

sistemas de información a menudo necesitan ser reemplazados con sistemas nuevos donde el flujo de transacciones entre sistemas ingresen estos datos sin la intervención de los humanos. Por ejemplo, una compañía puede haber recibido previamente los pedidos vía fax y capturados manualmente en el sistema de administración de pedidos y en el sistema de control de producción. Ahora, las compañías pueden recibir pedidos a través de un sitio Web y automáticamente ingresar la información en los otros sistemas relacionados tales como el de producción.

Una integración ideal debe satisfacer los siguientes requerimientos:

- No debe ser caro, y con un rápido retorno de inversión.
- Fácil de aprender y de administrar.
- Escalable, confiable, altamente disponible, tolerante a fallos y seguro.
- Flexible y fácilmente configurable.

Además la integración puede ser implementada en diferentes capas de la tecnología como por ejemplo [12]:

- **Integración de interfaz de usuario.** Se refiere a como se integra el conjunto completo de aplicaciones y servicios al cual un usuario determinado accede, para proveer consistencias y eficiencia.
- **Integración de datos.** Se enfoca en la integración a nivel de datos, a menudo para sincronizar el contenido de varias bases de datos. Los problemas claves encontrados envuelven la reconciliación de los esquemas de las bases de datos.
- **Integración de mensajes.** Se enfoca en la integración para intercambiar mensajes entre aplicaciones; los mensajes usualmente representan transacciones que han ocurrido en una aplicación y necesita ser propagada a otras aplicaciones. Los problemas claves encontradas en este tipo de integración la transformación de los diferentes formatos de mensaje entre diferentes aplicaciones.
- **Integración de componentes.** Se enfoca en la integración de sistema que usan diferentes componentes tecnológicos (CORBA, .NET o J2EE).
- **Integración de aplicaciones.** Se enfoca en la integración de aplicaciones usando sus APIs publicadas, modelos de objetos, formatos de mensajes, esquemas de bases de datos. Involucran los anteriores tipos de integración antes mencionados. La conectividad de aplicaciones es un estilo de integración involucrado con todos los tipos de conectividad relacionados con la arquitectura.
- **Integración de servicios.** Se enfoca en la creación de servicios de negocio que no están atados a una simple base de datos, modelo de componentes o aplicación, y usan estos servicios como bloques para la integración de sistemas.
- **Integración de procesos.** Se enfoca en la creación de nuevos procesos de negocios para la integración de componentes, aplicaciones y servicios. Esta forma de integración define explícitamente y administra los procesos de negocio separadamente de alguna aplicación específica.

- **Integración B2B.** Se enfoca en la automatización de procesos de negocio y servicios de negocios claves entre dos o más compañías.

Finalmente, un requerimiento para el ambiente de desarrollo de aplicaciones es que tenga en cuenta todos los estilos y niveles de integración posibles en su organización permitiendo su desarrollo y despliegue. Debe incluir una metodología como se diseñan y construyen componentes, para facilitar reutilización, eliminar redundancia y simplificar las pruebas, la instalación el mantenimiento.

Todos los tipos de integración mencionados están presentes en una organización, aun cuando en ocasiones son simplificados o no claramente definidos. Todos deben ser considerados al embarcarse en una nueva estructura arquitectónica.

Aún en un ambiente de información simple, con alcance limitado en la conectividad, las funciones de integración dentro de la estructura deben ser provistas por servicios, en vez de ser desempeñadas directamente por las aplicaciones. En esa forma se asegura que la estructura sobrevivirá al crecimiento y a los cambios que toda organización experimenta.

3.2. Integración e interoperabilidad usando Servicios Web

Con la llegada de estándares de la industria como por ejemplo Servicios Web, SOA está impulsando un cambio de paradigma en algunas áreas, incluyendo la integración de aplicaciones empresariales.

La arquitectura de integración orientada a servicios consiste en integrar entidades informáticas usando interacciones de servicios. Este modelo basado en los servicios se refiere a los problemas de integración de sistemas heterogéneos inflexibles y anticuados permitiendo que las organizaciones de tecnologías de información ofrezcan la funcionalidad bloqueadas de aplicaciones existentes como servicios reutilizables.

Construir una solución de integración que se aplique a la empresa es una tarea difícil. Estas soluciones necesitan integrar sistemas empresariales múltiples que no están creados para trabajar juntos.

La integración de estos sistemas es difícil por varios motivos. Esto incluye la heterogeneidad de plataformas y lenguajes de programación, la diversidad y complejidad de cada sistema empresarial individual y la dificultad de comprensión de los requerimientos para la solución integrada resultante los arquitectos de software realizan un sin fin de tareas cruciales durante el diseño de aplicaciones empresariales integradas. Entre ellas se incluyen las siguientes:

- Colaborar en la comprensión de requerimientos funcionales y de calidad para las aplicaciones integradas.
- Crear el proyecto de arquitectura inicial para las aplicaciones integradas.
- Confirmar que la combinación de la arquitectura y la tecnología de integración que se utilizan para construir la aplicación para toda la

empresa pueden ser exitosas antes de realizar una gran inversión para su implementación.

Una vez que los servicios Web han sido definidos, es necesario categorizarlos y almacenarlos en un registro de servicios para que los consumidores puedan localizarlos.

En particular las tecnologías de servicios Web:

- Proveen una nueva manera para permitir a las aplicaciones existentes y nuevas interoperar entre ellas.
- Suportan la capacidad para crear aplicaciones compuestas para que puedan combinarse rápida y fácilmente.
- Realizan de la manera más fácil, la combinación de varias fuentes de información usando XML como el estándar de datos.
- Reducen la necesidad de tener empleados altamente entrenados para interactuar con sistemas de información para generar reportes, y actualizar documentos importantes.

La siguiente figura ilustra el problema cuando múltiples sistemas con distintos ambientes de aplicación interoperan. Una vez más, podemos ver el rol que juega el contrato de servicio (WSDL) en la definición de servicios provistas por cada sistema.

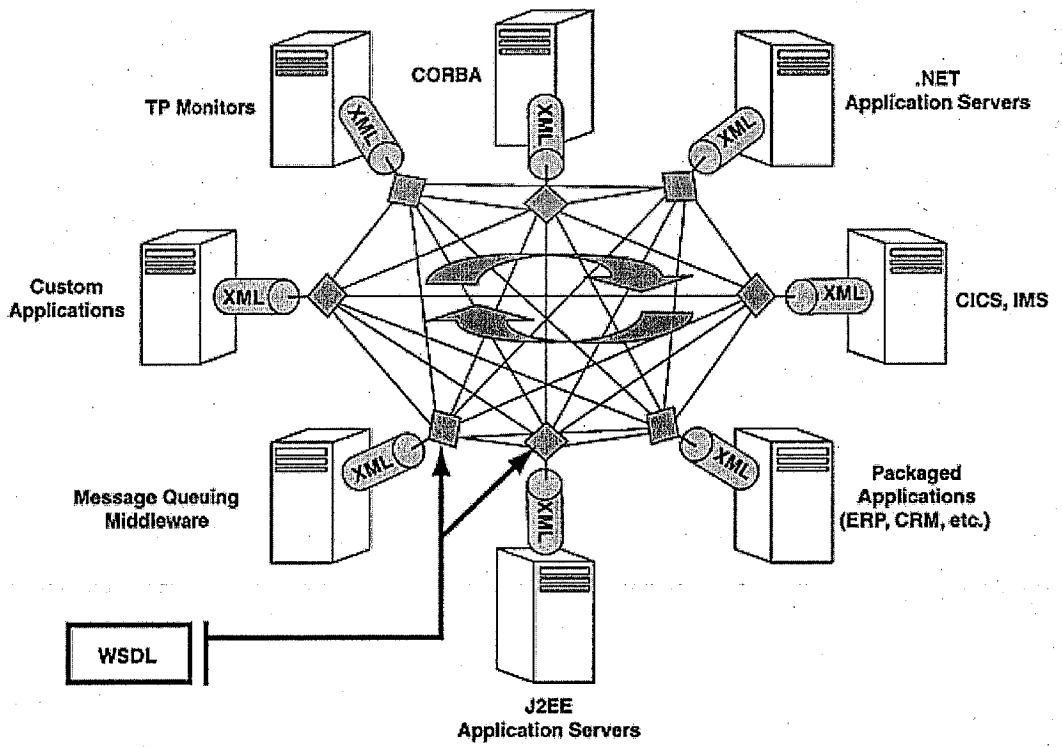


Figura 5. Múltiples tecnologías de servicios Web [12]

Cuando se usa los servicios Web como la base para la integración, como se muestra en la figura anterior, los datos son convertidos en un formato canónico

entendido por todos (en este caso XML), y los mensajes son pasados en un formato estándar de un punto a otro. Cada punto es responsable para transformar sus tipos de datos nativos en formatos cuando reciben un mensaje.

Sin embargo, hay casos claramente en el que el enfoque de mensaje canónico no tiene sentido. Esto es cuando las tecnologías en ambos lados de la comunicación son similares. Por ejemplo no tiene sentido la conversión de estructura de datos CORBA a XML y entonces regresar otras ves cuando el que envía y el que recibe ambos usando CORBA y están corriendo dentro de un firewall.

Las descripciones actuales de servicios las representan los WSDL, los cuales describen la manera de acceder a la funcionalidad. Una vez que se decide cuál funcionalidad se desea, el WSDL describe como decodificar el mensaje, cómo enviarlo, cómo asegurarlo, cómo autenticar al cliente y todo lo que se necesite para describir el proceso de un mensaje (no describe qué solicitar). Para cualquier cliente o consumidor de servicio dedicado que ya sabe qué solicitar, esto es genial y ya que la infraestructura y la descripción se basan en estándares, el código es reutilizable y muchas tecnologías pueden interactuar. Esto significa que se puede crear comunicaciones entre muchas partes basándose en una sola infraestructura.

Sin embargo, esto implica que lo que le solicita esté codificado en la aplicación. Cuando las empresas desean ser flexibles, pueden querer acceder a servicios para conseguir información, agregarla, y reordenarla, o pueden desear cambiar el flujo de procesamiento. Actualmente, para lograr esto, necesitan desarrolladores. Esto puede tornar costoso el cambio, y en muchos casos demasiado costoso. Las empresas que se encuentran frente a este problema prefieren tener analistas comerciales y usuarios inclusive, para que realicen los cambios. Esto lograría que los cambios sean más económicos, acortarían el ciclo de cambio y lo agilizarían.

Si nuestro software pudiera comunicar esto a la gente de negocios, ellos podrían definir nuevos agregados de información y cambiar o crear un flujo de procesamiento sin la ayuda de los desarrolladores.

La integración orientada a servicios es la aplicación de servicios Web para resolver los problemas de integración e interoperabilidad. Este tipo de integración promete a las organizaciones un mejor trabajo y una mejoría de resultados a largo plazo.

3.3. Enfoque de integración basado en servicios.

A diferencia de la tradicional Integración de aplicaciones empresariales, las características principales del enfoque basado en servicios para la integración son [13]:

- Interfaz estándar bien definida. Se proporciona a los consumidores acceso sistemático y de fácil comprensión para los servicios subyacentes.

- Opacidad. La tecnología y ubicación de la aplicación que proporciona la funcionalidad están ocultas detrás de la interfaz. De hecho, no existe la necesidad de un proveedor de servicios fijos.
- Flexibilidad. Tanto el proveedor de servicios como el consumidor pueden cambiar, la descripción del servicio del servicio es la única constante. Siempre que el proveedor y el consumidor sigan fieles a la descripción del servicio, las aplicaciones seguirán funcionando.

Este enfoque de integración orientada a servicios consiste en:

- La refinación de modelos de datos para que adecuen a la integración.
- La definición de contratos formales de servicio mas allá de que puedan ser definidas en WSDL para incluir las reglas negocio, políticas de seguridad y políticas de administración. Cuando sea posible estas políticas son usadas para definir y comunicar las decisiones.
- La definición de transformaciones de datos que son mapeados entre los diferentes modelos de datos.
- La configuración de los ambientes de ejecución para los servicios Web para asegurar el cumplimiento de los requerimientos.

Además para asegurar el éxito de la integración basada en servicios, es necesario que la tecnología de integración posea las siguientes cualidades:

- Escalabilidad
- Alto rendimiento
- Seguridad
- Manejabilidad

Como podemos observar uno de los puntos principales se concentra en utilizar los estándares de la industria como los Servicios Web para permitir la entrega del mensaje real y otro diversos servicios avanzados, evitando los problemas de las tecnologías tradicionales, es decir, el uso de protocolos de propiedad exclusiva para intercambio de mensajes. De esta manera, la integración basada en servicios es un modelo de diseño que asegura la interoperabilidad e integración verdadera en cualquier contexto empresarial heterogéneo.

Existen diversas implementaciones variadas de tecnologías de integración que proporcionan las funcionalidades mencionadas anteriormente. Estas abarcan desde las tecnologías tradicionales que han incluido prestaciones de Servicios Web, hasta implementaciones nuevas de marcas que poseen soporte de Servicios de Web inherentes.

Algunas ventajas de la integración orientada a servicios son:

- La creación de datos formales reutilizables, servicio, y modelo de procesos que son ampliamente aplicables en otros dominios.
- Una plataforma de servicios Web es establecida y crea una capa de abstracción que reduce los candados y simplifica la migración y consolidación en el futuro.
- La plataforma de servicios Web provee atributos de calidad tales como la seguridad, confiabilidad, entre otros.

- El código de la aplicación no necesitará implementar los atributos de calidad tales como seguridad y confiabilidad ya que los servicios Web se encargan de proveerlos.

Algunas limitaciones de la integración orientada a servicios son:

- Requiere de un grupo de arquitectos de software dedicados y con habilidades.
- Requiere el compromiso de los administradores de negocios y administradores técnicos.
- Requiere una cultura que promueva la estrategia para implementar esta tecnología con la finalidad de mejorar la organización alcanzando grandes valores para el negocio.

4. Etapas de adopción de SOA

El impacto inicial que los servicios Web han tenido en los últimos años ha propiciado que muchas compañías se encuentren interesadas en adoptar SOA para aprovechar las oportunidades de integración que esta provee.

Hoy en día no hay en la industria una metodología a seguir para que las organizaciones inicien una adopción de este tipo de arquitecturas. En este documento se propone una manera de iniciar una adopción incremental de SOA a través del seguimiento de 4 etapas:

1. Experimentar con Servicios Web.
2. Adaptar sistemas existentes para que usen servicios Web.
3. Remover dependencias entre sistemas.
4. Establecer internamente una arquitectura orientada a servicios.

Estas etapas son especificadas en las secciones siguientes:

4.1. Etapa 1: Experimentando con servicios Web

La mejor manera de iniciar con Servicios Web es empezar con pequeños proyectos ya que tienen una alta oportunidad de éxito. Algunas maneras de empezar a experimentar con los servicios Web son:

- **Uso de un servicio externo.** Probablemente la forma más básica para empezar a experimentar es el uso de un servicio externo. Existen múltiples servicios externos simples que podemos elegir y encontrar[1]. .El uso un servicio externo ayuda a las organizaciones a probar los servicios Web sin mucho desgaste de tiempo y dinero. Esto nos dará una idea de cómo los servicios Web trabajan y nos dará la pauta para iniciar el desarrollo de un servicio interno.
- **Desarrollo de un servicio Web.** Se puede empezar desarrollando un servicio Web que comunique dos capas de aplicación en una arquitectura distribuida. Por ejemplo si se quiere comunicar las capas de negocio y de presentación a través de un servicio Web, entonces el servicio deberá proporcionar toda la funcionalidad que se necesitará las aplicaciones cliente para acceder a la lógica de negocio de la aplicación.

4.2. Etapa 2: Adaptando sistemas existentes para el uso de Servicios Web.

En una organización seguramente se necesitará comunicar dos sistemas internos para que intercambien información. Por ejemplo si contamos con los sistemas A y B; ambos pueden permitir a los usuarios ingresar la información de clientes; si uno de los sistemas actualiza la información de un cliente, el otro sistema debería también actualizarse. Otro ejemplo es que A solo se encargue de la información de los clientes, pero B necesite comunicarse con A para validar la información de un cliente. Estos son casos donde el uso de servicios

[1] www.xmethods.net

Web es una buena alternativa para comunicar sistemas, especialmente cuando estos sistemas han sido desarrollados en tecnologías diferentes.

4.3. Etapa 3: Removiendo dependencias entre sistemas.

Eliminar las dependencias entre sistemas nos dará más flexibilidad para la integración de aplicaciones. Las principales actividades para esta etapa son:

- Identificar las aplicaciones que mantienen datos redundantes.
- Factorizar los datos redundantes dentro de bases de datos maestras y conectar las aplicaciones a adaptadores de servicios Web.
- Identificar que aplicaciones que usan los mismos datos de la base de datos compartida.
- Determinar si estas aplicaciones deberían estar separadas.
- Para estas aplicaciones que podrían ser separadas, determinar si mantienen sus propios estados.
- Para estas aplicaciones que mantienen sus propios estados, factorizar los datos en base de datos maestra y conectar a los adaptadores de servicios Web.
- Se deben separar las interfaces de usuario de la lógica de negocio para incrementar las posibilidades de reutilización y sobre todo facilitar la modificación.

4.4. Etapa 4: Establecer un Arquitectura Orientada a Servicios Interna.

En este punto, la organización deberá tener la experiencia necesaria respecto a servicios Web y podrán iniciar una implementación interna de una arquitectura orientada a servicios. También se necesitará tomar algunas consideraciones:

- Determinar si se debería usar datos centralizados, procesos distribuidos o combinar ambos enfoques.
- Decidir si un data warehouse, business intelligence, o agentes tienen sentido en la arquitectura.

Hay algunas consideraciones de diseño cuando se establece una arquitectura orientada a servicios:

- Cada servicio debe ser capaz de recibir mensajes múltiples veces con ningún efecto adverso. Por ejemplo, un servicio puede recibir actualizaciones de datos más de una vez sin afectar los datos.
- , los mensajes deberían ser enviados con un servicio de alta capacidad y velocidad.

Es importante elegir al personal correcto para hacer esta experimentación. Francamente, en muchas situaciones, es un riesgo involucrar a gente que nunca ha expresado mucho interés en probar algo nuevo. Es por esto, que la elección debe involucrar a gente que le guste experimentar nuevas tecnologías, ya que será una mejor opción. Para muchas organizaciones podría ser mejor contratar a alguien quie tenga conocimiento de Servicios Web y arquitecturas orientadas a servicios y quien sea quien guíe a otros desarrolladores a realizar esta etapa de experimentación.

5. Construyendo un servicio Web, paso a paso.

En esta sección mostraremos la puesta en marcha de la construcción de un servicio Web para cumplir el objetivo de integrar dos aplicaciones desarrolladas en plataformas distintas, como son el caso de .NET y Borland C++.

5.1.1. Descripción del Problema.

La gerencia de desarrollo de software del Centro de Investigación en Matemáticas (CIMAT) pretende crear una aplicación que interprete y reconozca las palabras escritas a mano en el pizarrón multimedia de esta institución.

El software que actualmente se ha desarrollado para el pizarrón esta implementado en la plataforma de Borland C++ Builder. Actualmente Microsoft cuenta con una librería de código para el reconocimiento de escritura a mano para la plataforma Tablet PC que la gerencia de desarrollo del CIMAT ha decidido evaluar para decidir si la funcionalidad ofrecida por esta librería se incorpora en el desarrollo del software del pizarrón para cumplir con el objetivo de reconocimiento de escritura a mano. La figura 6 muestra una aplicación de ejemplo que usa la librería de reconocimiento de Microsoft.

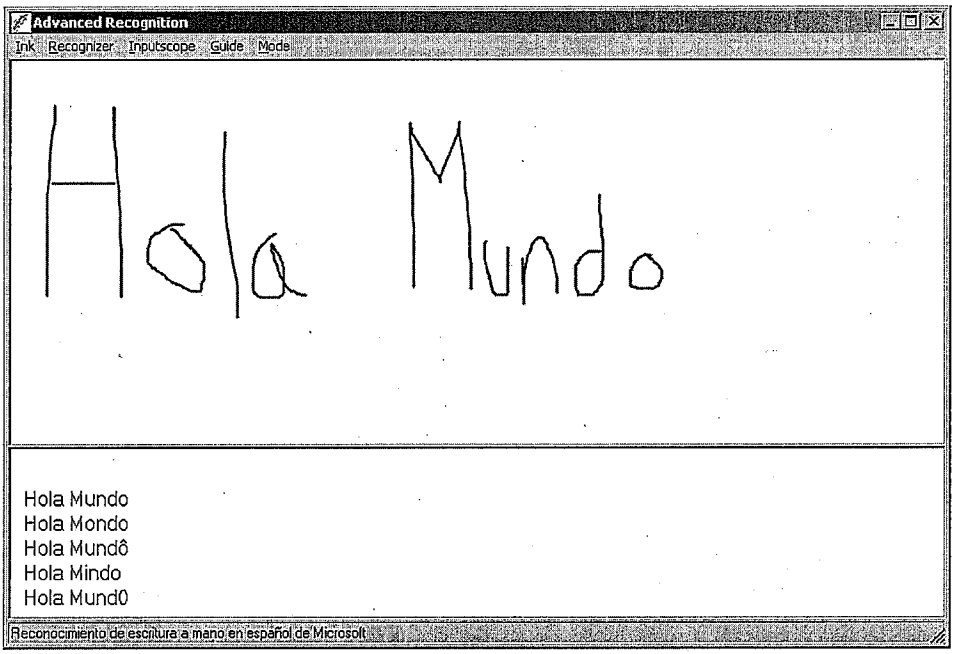


Figura 6. Ejemplo de reconocimiento de escritura a mano.

Sin embargo el uso de esta librería de código tiene algunos inconvenientes de implementación: requiere que el sistema operativo donde se ejecuta sea Windows XP; el entorno de desarrollo donde se utilice sea Visual Basic 6.0 o Visual Studio .NET (Versión 2002 o Superior).

Como podemos darnos cuenta de entrada la aplicación de pizarrón y la librería de Microsoft se ejecutan en plataformas tecnológicas distintas, como son:

Borland c++ Builder y .NET respectivamente. Entonces el reto es integrar estos dos elementos de software.

5.1.2. Solución Propuesta.

La propuesta entonces, es implementar un servicio Web que proporcione a las aplicaciones clientes (de distintas plataformas) la funcionalidad del reconocedor de escritura de mano.

La arquitectura propuesta se muestra en la siguiente figura.

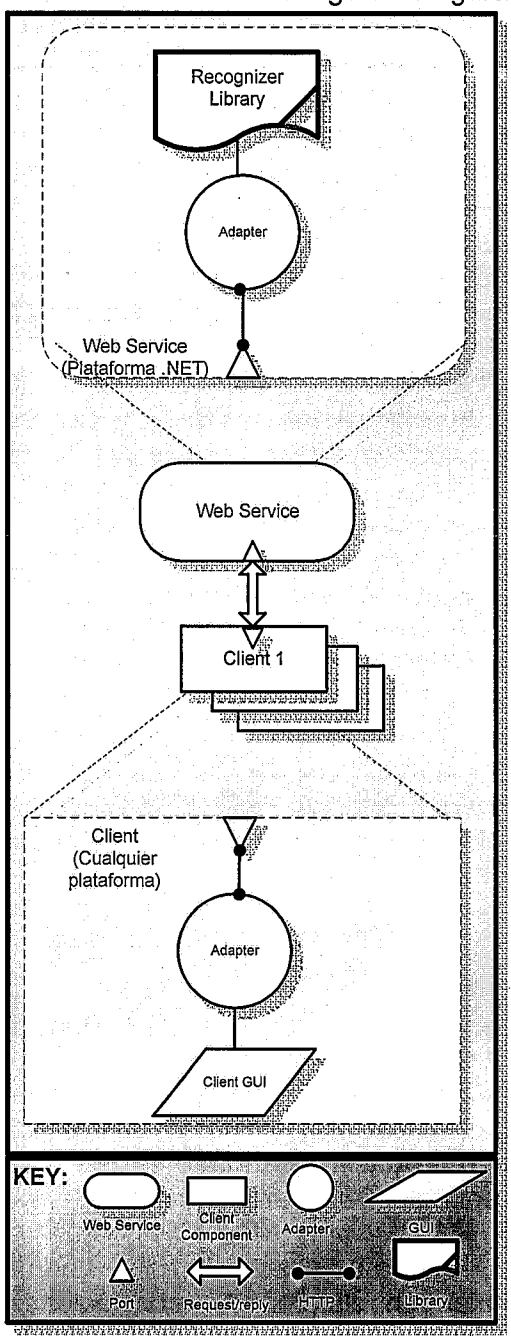


Figura 7: Arquitectura Propuesta.

Los elementos de la arquitectura son:

- **Web Service.** Representa al servicio Web que proporcionara la funcionalidad del reconocedor.
- **Adapter.** Componente que se encargará de transformar los tipos de datos internos a tipos de datos del contrato del servicio y viceversa.
- **Recognizer Library.** Representa al componente reconocedor de escritura a mano, en este caso la librería creada por Microsoft.
- **Cliente.** Representa una aplicación (en cualquier plataforma) que consumirá el servicio Web.
- **Port.** El puerto en el que se comunicará el cliente con el servicio Web.
- **Client GUI.** Representa la interfaz de usuario de la aplicación cliente.

En las secciones siguientes, se especificarán los pasos para crear esta arquitectura.

5.1.3. Paso 1: Diseñar de los mensajes y tipos de datos.

En este paso se diseñan conceptualmente los mensajes y tipos de datos que el servicio Web publicará en el contrato.

Primero diseñamos los tipos de datos que serán la entrada y la salida de los mensajes de los servicios Web. Ver figura.

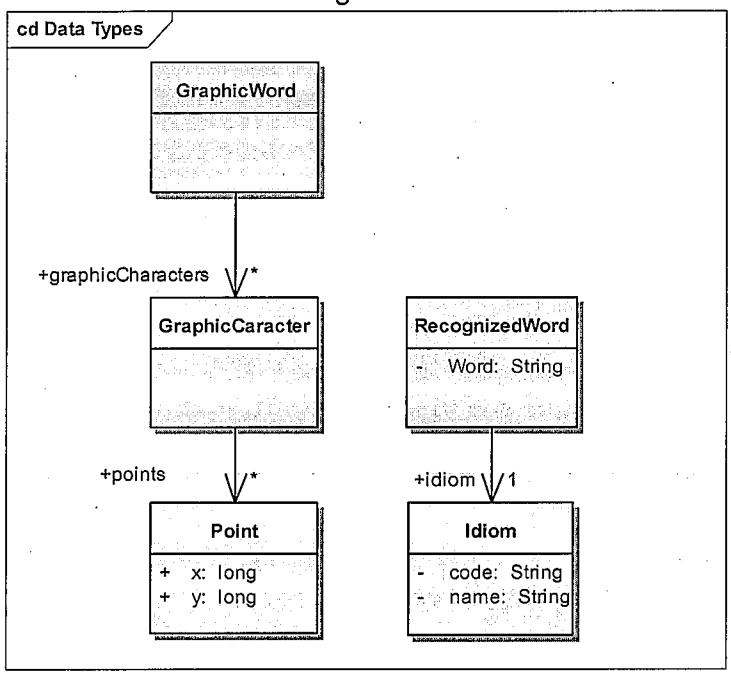


Figura 8. Tipos de datos del contrato.

Los tipos de datos son:

- **Point**. Representa un punto en el trazo de la letra dibujada a mano.
- **GraphicCaracter**. Representa un arreglo de Point que forma el trazo de la letra dibujada.
- **GraphicWord**. Representa un arreglo de GraphicCaracter que forma las letras dibujadas.
- **Idiom**. Representa el idioma al que será traducido la palabra dibujada a mano.
- **RecognizedWord**. Representa la palabra que se ha reconocido al procesar el GraphicWord en el servicio reconocedor.

Los mensajes XML representan las operaciones que el servicio Web publica para que sean accesibles. Los mensajes no contienen lógica de implementación y solamente documentan el nombre de una operación y sus tipos de entradas y salidas. Deben ser diseñados en conjunto con los esquemas XSD.

La Figura 9 muestra el diseño del servicio Web y las operaciones (mensajes) que serán accesibles para los clientes. Para que el servicio este presente bajo acoplamiento se puede implementar un patrón de fachada para que se separe del servicio Web la lógica interna del reconocedor.

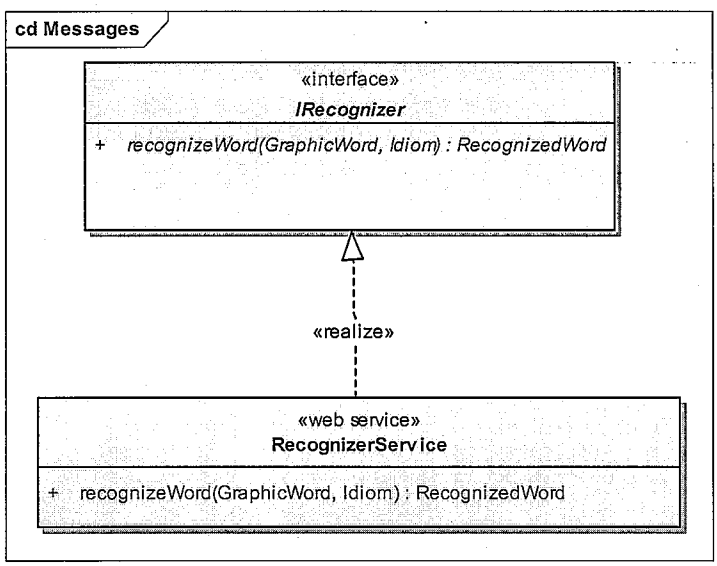


Figura 9. Operación del Servicio Web.

Como podemos ver el Servicio expondrá una operación llamada *recognizeWord.* Tal operación recibe como entrada un *GraphicWord* y el Idioma en que la palabra será reconocida. El tipo de retorno de la operación es un *RecognizedWord* que tendrá la palabra reconocida en el Idioma especificado.

5.1.4. Paso 2: Construir el esquema XML para los tipos de datos.

Los diagramas UML generados en el paso 1 son la base conceptual para los esquemas de tipos de datos.

En este paso podemos usar una herramienta de diseño de XML para construir el esquema para los tipos de datos del servicio Web. Podemos usar un diseñador que sea el más confortable para nosotros. Para este ejemplo usaremos el diseñador de esquemas XML de Visual Studio .NET.

La figura muestra la vista del diseñador de Visual Studio .NET

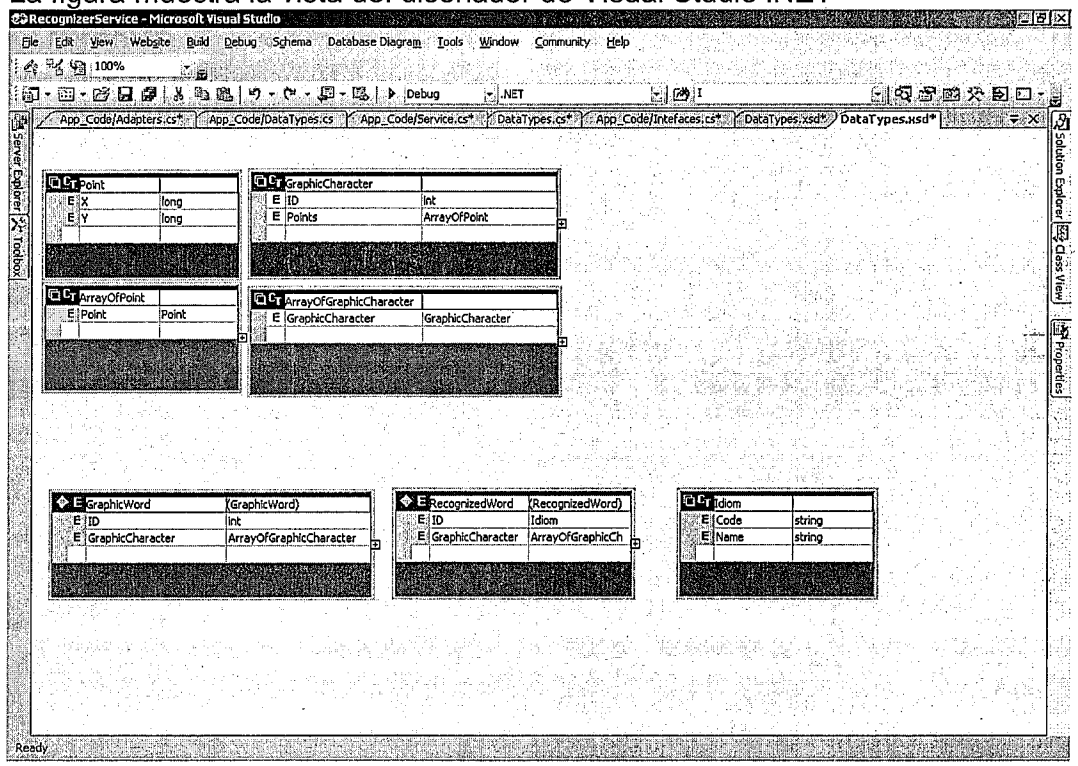


Figura 10: Diseñador de XML de Visual Studio .NET mostrando el esquema XSD de los tipos de datos del servicio.

Internamente el diseñador genera la siguiente estructura en XML.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="DataTypes" targetNamespace="http://tempuri.org/DataTypes.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/DataTypes.xsd"
xmlns:mstns="http://tempuri.org/DataTypes.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Idiom">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="1" name="Code" type="xs:string" />
      <xs:element minOccurs="0" maxOccurs="1" name="Name" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="GraphicWord">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="1" name="ID" type="xs:int" />
        <xs:element minOccurs="0" maxOccurs="1" name="GraphicCharacter"
type="ArrayOfGraphicCharacter" />
      </xs:sequence>
    </xs:complexType>
```

```
  </xs:element>
  <xs:element name="RecognizedWord">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="ID" type="Idiom" />
        <xs:element minOccurs="0" maxOccurs="1" name="GraphicCharacter"
type="ArrayOfGraphicCharacter" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="ArrayOfGraphicCharacter">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="GraphicCharacter"
nillable="true" type="GraphicCharacter" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="GraphicCharacter">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="ID" type="xs:int" />
      <xs:element minOccurs="0" maxOccurs="1" name="Points" type="ArrayOfPoint" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ArrayOfPoint">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Point" nillable="true"
type="Point" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Point">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="X" type="xs:long" />
      <xs:element minOccurs="1" maxOccurs="1" name="Y" type="xs:long" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Información sobre la creación de esquemas XML y el Diseñador de .NET[2].

5.1.5. Paso 3: Crear la clase de las definiciones de los tipos de datos del servicio Web partir del esquema XML.

Para crear la clase de las definiciones de los tipos de datos podemos hacer uso de herramientas que construyen clases a partir de esquemas XML. El framework de .NET proporciona una herramienta en línea de comandos llamada xsd.exe para la generación de las definiciones de los tipos de datos a partir de un esquema XML. La figura muestra el uso de la herramienta xsd.exe.

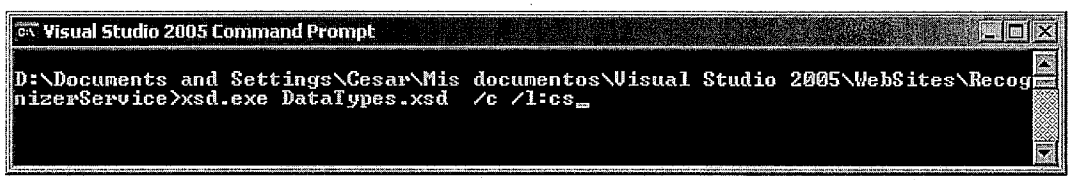


Figura 11. Usando xsd.exe para generar las clases de los tipos de datos

Después de la generación, las clases de los tipos de datos serian:

```
 [System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
```

[2] http://msdn.microsoft.com/library/SPA/vbcon/html/vboriCreatingSchemas.asp

```
[System.Xml.Serialization.XmlRootAttribute(Namespace="http://tempuri.org/DataTypes.xsd",
IsNullable=false)]
public partial class Idiom {
    private short codeField;

    private string nameField;

    public short Code {
        get {
            return this.codeField;
        }
        set {
            this.codeField = value;
        }
    }

    public string Name {
        get {
            return this.nameField;
        }
        set {
            this.nameField = value;
        }
    }
}
```

```
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://tempuri.org/DataTypes.xsd")
]
public partial class Point {

    private long xField;

    private long yField;

    public long X {
        get {
            return this.xField;
        }
        set {
            this.xField = value;
        }
    }

    public long Y {
        get {
            return this.yField;
        }
        set {
            this.yField = value;
        }
    }
}
```

```
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://tempuri.org/DataTypes.xsd")
]
public partial class GraphicCharacter {

    private int idField;

    private Point[] pointsField;

    public int ID {
        get {
            return this.idField;
        }
        set {
            this.idField = value;
```

```
        }
    }

    public Point[] Points {
        get {
            return this.pointsField;
        }
        set {
            this.pointsField = value;
        }
    }
}
```

```
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
[System.Xml.Serialization.XmlRootAttribute(Namespace="http://tempuri.org/DataTypes.xsd",
IsNullable=false)]
public partial class GraphicWord {

    private int idField;

    private GraphicCharacter[] graphicCharacterField;

    public int ID {
        get {
            return this.idField;
        }
        set {
            this.idField = value;
        }
    }

    public GraphicCharacter[] GraphicCharacters {
        get {
            return this.graphicCharacterField;
        }
        set {
            this.graphicCharacterField = value;
        }
    }
}
```

```
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
[System.Xml.Serialization.XmlRootAttribute(Namespace="http://tempuri.org/DataTypes.xsd",
IsNullable=false)]
public partial class RecognizedWord {

    private Idiom idiomField;

    private string wordField;

    public Idiom Idiom {
        get {
            return this.idiomField;
        }
        set {
            this.idiomField = value;
        }
    }

    public string Word {
        get {
            return this.wordField;
        }
        set {
            this.wordField = value;
        }
    }
```

```
}
```

Estas clases fueron generadas en el lenguaje de programación C# de la plataforma .NET.

5.1.6. Paso 4: Crear la Interfase y el Adaptador del servicio Web.

Para esto nos basamos en el diagrama UML de la definición de la interfase del servicio Web del paso 1. La interfase entonces sería:

```
namespace Interfaces
{
    public interface IRecognizer
    {
        RecognizedWord recognizeWord(GraphicWord graphicWord, Idiom idiom);
    }
}
```

Ahora continuamos construyendo el adaptador del servicio, quien será el encargado de implementar la operación de la interfase y además su principal función es transformar o adaptar los tipos de datos del servicio Web a tipos de datos internos y viceversa.

Para el caso del reconocedor de escritura a mano el Adaptador debe conocer y referenciar la librería de este. La clase del adaptador sería:

```
namespace Adapters
{
    using Interfaces;
    using System.Collections;
    public class RecognizerAdapter : IRecognizer
    {
        public RecognizedWord recognizeWord(GraphicWord graphicWord, Idiom idiom)
        {
            ArrayList internalPointsList = new ArrayList();

            //Se transforma cada caracter de la palabra
            foreach (GraphicCharacter character in graphicWord.GraphicCharacters)
            {
                //Se transforma cada punto del caracter
                foreach (Point point in character.Points)
                {
                    //Se transforma el tipo de dato (Point) del servicio a un
System.Drawing.Point

                    System.Drawing.Point internalPoint = new System.Drawing.Point();
                    internalPoint.X = (int) point.X;
                    internalPoint.Y = (int) point.Y;

                    internalPointsList.Add(internalPoint);
                }
            }

            handwriting.Idiom internalIdiom = handwriting.Idiom.Spanish ;

            //Se tranforma el tipo de dato (Idiom) del servicio a un handwriting.Idiom
            switch (idiom.Name.ToLower())
            {
                case "spanish": internalIdiom = handwriting.Idiom.Spanish; break;
                case "english": internalIdiom = handwriting.Idiom.English; break;
                case "italian": internalIdiom = handwriting.Idiom.Italian; break;
                case "french": internalIdiom = handwriting.Idiom.French;   break;
            }
```

```
            string word;
            System.Drawing.Point[] internalPoints = (System.Drawing.Point[])
                internalPointsList.ToArray(typeof(System.Drawing.Point));

            //Se invoca el reconocedor
            word = handwriting.Recognizer.Recognize(internalPoints,
                                                    internalIdiom);

            //Se transforma el resultado
            RecognizedWord recognizedWord = new RecognizedWord();

            recognizedWord.Idiom = idiom;
            recognizedWord.Word = word;

            return recognizedWord;
        }
    }
}
```

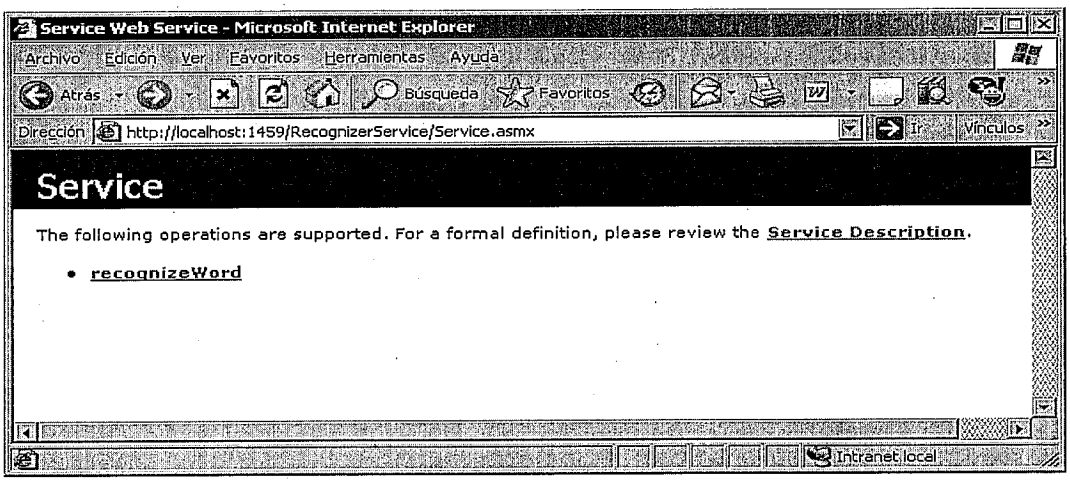
5.1.7. Paso 5: Implementar el Servicio Web.

Ahora que tenemos listos los tipos de datos, la interfase de servicio y el adaptador podemos construir el servicio Web que será publicado a las aplicaciones clientes para que estos consuman la funcionalidad del reconocedor. El código del servicio Web es:

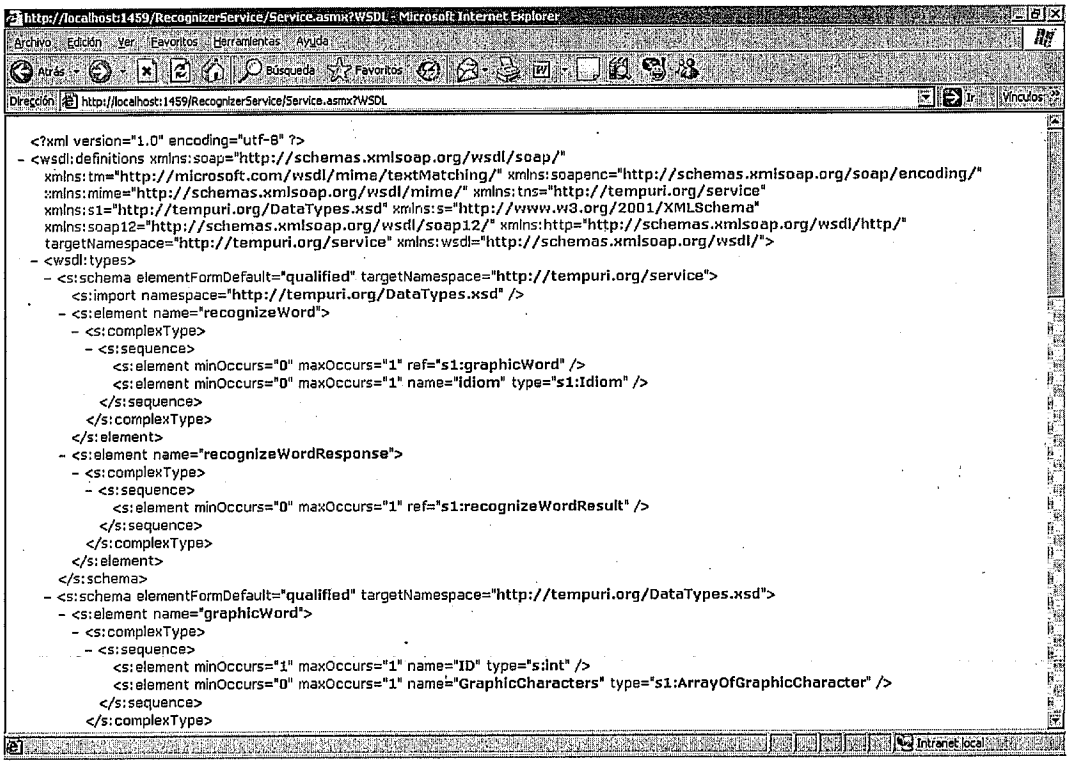
```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using Interfaces;
using Adapters;

[WebService(Namespace = "http://tempuri.org/service")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService, IRecognizer
{
    [WebMethod]
    public RecognizedWord recognizeWord(GraphicWord graphicWord,
                                        Idiom idiom)
    {
        IRecognizer adapter = new RecognizerAdapter();

        return adapter.recognizeWord(graphicWord, idiom);
    }
}
```

5.1.8. Paso 6: Ejecutar el Servicio Web.

Si ejecutamos este servicio Web desde Visual Studio .NET veremos la siguiente pantalla:



La pantalla muestra dirección de un servicio Web en el cual podemos consultar las operaciones que están disponibles. También podemos ver la descripción del servicio Web misma que representa el contrato para los clientes. En este contrato se definen los tipos de datos y mensajes necesarios para consumir el servicio Web. Este contrato se muestra en la siguiente imagen.

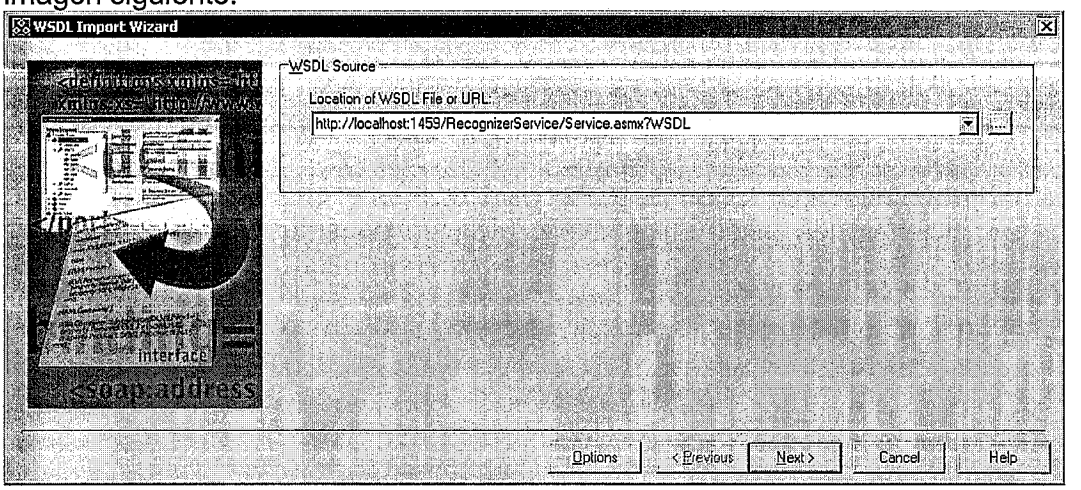



```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://tempuri.org/service"
    xmlns:s1="http://tempuri.org/DataTypes.xsd" xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    targetNamespace="http://tempuri.org/service" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/service">
        <s:import namespace="http://tempuri.org/DataTypes.xsd" />
      - <s:element name="recognizeWord">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" ref="s1:graphicWord" />
              <s:element minOccurs="0" maxOccurs="1" name="idiom" type="s1:Idiom" />
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="recognizeWordResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" ref="s1:recognizeWordResult" />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:schema>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/DataTypes.xsd">
      - <s:element name="graphicWord">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="1" maxOccurs="1" name="ID" type="s:int" />
              <s:element minOccurs="0" maxOccurs="1" name="GraphicCharacters" type="s1:ArrayOfGraphicCharacter" />
            </s:sequence>
          </s:complexType>
```

5.1.9. Paso 7: Consumir el Servicio Web.

Para este paso vamos a consumir el servicio Web desde una aplicación cliente desarrollada en la plataforma de Borland C++ Builder;

Para referenciar al servicio Web usamos el "WSDL Import Wizard" de la herramienta de desarrollo (C++ Builder 6.0). En este wizard necesitamos especificar la dirección del contrato del servicio Web[3], para este ejemplo. Ver imagen siguiente.



Este wizard generará un archivo de clases C++ con los tipos de datos especificados en el contrato. El archivo generado se verá:

```
#ifndef   ServiceH
#define   ServiceH

#include <System.hpp>
#include <InvokeRegistry.hpp>
#include <XSBuiltIns.hpp>
#include <SoapHTTPClient.hpp>


namespace NS_Service {

class  Point;
class  GraphicCharacter;
class  GraphicWord;
class  Idiom;
class  RecognizedWord;

class Point : public TRemotable {
private:
  __int64         FX;
  __int64         FY;
public:
__published:
  __property __int64             X = { read=FX, write=FX };
  __property __int64             Y = { read=FY, write=FY };
};

typedef DynamicArray<Point*>      ArrayOfPoint;   /* "http://tempuri.org/service" */
```

[3] http://localhost:1459/RecognizerService/Service.asmx?WSDL

```
class GraphicCharacter : public TRemotable {
private:
  int             FID;
  ArrayOfPoint    FPoints;
public:
  __fastcall ~GraphicCharacter();
__published:
  __property int                ID = { read=FID, write=FID };
  __property ArrayOfPoint     Points = { read=FPoints, write=FPoints };
};

typedef        DynamicArray<GraphicCharacter*>        ArrayOfGraphicCharacter;        /*
"http://tempuri.org/service" */

class GraphicWord : public TRemotable {
private:
  int             FID;
  ArrayOfGraphicCharacter FGraphicCharacters;
public:
  __fastcall ~GraphicWord();
__published:
  __property int                ID = { read=FID, write=FID };
  __property  ArrayOfGraphicCharacter  GraphicCharacters  =  {  read=FGraphicCharacters,
write=FGraphicCharacters };
};

class Idiom : public TRemotable {
private:
  Smallint        FCode;
  AnsiString      FName;
public:
__published:
  __property Smallint         Code = { read=FCode, write=FCode };
  __property AnsiString       Name = { read=FName, write=FName };
};

class RecognizedWord : public TRemotable {
private:
  Idiom*          FIdiom;
  AnsiString      FWord;
public:
  __fastcall ~RecognizedWord();
__published:
  __property Idiom*          Idiom = { read=FIdiom, write=FIdiom };
  __property AnsiString       Word = { read=FWord, write=FWord };
};

// ************************************************************************ //
// Namespace : http://tempuri.org/service
// soapAction: http://tempuri.org/service/recognizeWord
// transport : http://schemas.xmlsoap.org/soap/http
// binding   : ServiceSoap
// service   : Service
// port      : ServiceSoap
// URL       : http://localhost:1459/RecognizerService/Service.asmx
// ************************************************************************ //
__interface   INTERFACE_UUID("{7101851F-158C-647C-C884-A4B3FF7C3CF0}")   ServiceSoap   :
public IInvokable
{
public:
  virtual  RecognizedWord*  recognizeWord(const  GraphicWord*  graphicWord,  const  Idiom*
idiom) = 0;
};
typedef DelphiInterface<ServiceSoap> _di_ServiceSoap;

_di_ServiceSoap GetServiceSoap(bool useWSDL=false, AnsiString addr="");


#endif // __Service_h__

};     // NS_Service
```

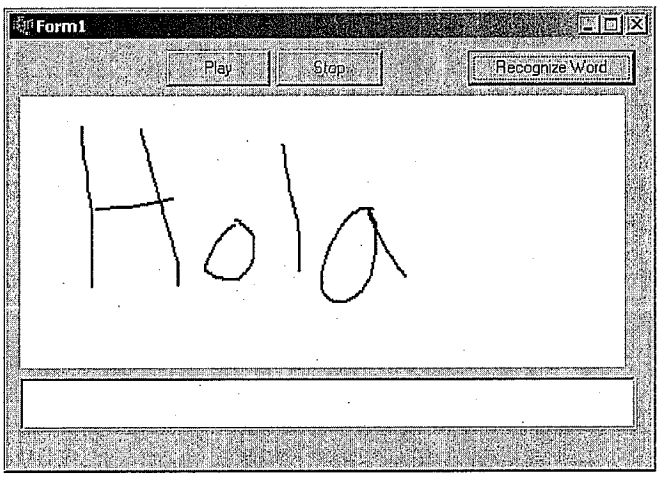
```
#if !defined(NO_IMPLICIT_NAMESPACE_USE)
using  namespace NS_Service;
#endif
```

Ahora lo que sigue es usar esta clase en nuestra aplicación C++ e invocar al servicio. Para esto vamos a utilizar un adaptador de este lado para que transforme los tipos de datos del servicio Web a tipos de datos internos a la plataforma.
El código del adaptador se muestra a continuación:

```
#include <vector.h>
#include "HandWriting.h"
#include "Service.h"
#include <vcl>
namespace Adapters {
   class RecognizerAdapter
   {
   private:
      AnsiString defWSDL;
      _di_ServiceSoap service;

   public:
      RecognizerAdapter( AnsiString URL_WSDL );

      HandWriting::RecognizedWord
      RecognizerAdapter::RecognizeWord( HandWriting::GraphicWord internalWord,
                                        HandWriting::Idiom internalIdiom );

   };
}
#endif

#include "Adapters.h"
using namespace Adapters;
RecognizerAdapter::RecognizerAdapter( AnsiString URL_WSDL )
{
   this->defWSDL = URL_WSDL;
   this->service = GetServiceSoap(true, URL_WSDL);
}
HandWriting::RecognizedWord
   RecognizerAdapter::RecognizeWord(       HandWriting::GraphicWord       internalWord,
HandWriting::Idiom internalIdiom )
{

      NS_Service::Idiom * idiom = new NS_Service::Idiom();
      idiom->Code = internalIdiom.Code ;
      idiom->Name = internalIdiom.Name ;


      NS_Service::GraphicWord * wordService = new NS_Service::GraphicWord();
      wordService->GraphicCharacters.Length = internalWord.Characters.size() ;

   for( unsigned int i = 0; i < internalWord.Characters.size(); i++)
   {
      NS_Service::GraphicCharacter * character = new NS_Service::GraphicCharacter();

      character->Points.Length = internalWord.Characters[i]->Points.size() ;

      for( unsigned int j = 0; j < internalWord.Characters[i]->Points.size(); j++)
      {
         NS_Service::Point * point = new NS_Service::Point ;
         point->X = internalWord.Characters[i]->Points[j]->X;
         point->Y = internalWord.Characters[i]->Points[j]->Y;
         character->Points[j] = point;
      }

      wordService->GraphicCharacters[i] = character;
   }

      NS_Service::RecognizedWord * recognizedWord = new NS_Service::RecognizedWord();
```

```
      recognizedWord = service->recognizeWord(wordService,idiom);

      HandWriting::RecognizedWord internalRecognizedWord;

      internalRecognizedWord.idiom.Code = recognizedWord->Idiom->Code ;
      internalRecognizedWord.Word = recognizedWord->Word ;

   return internalRecognizedWord;
}
```

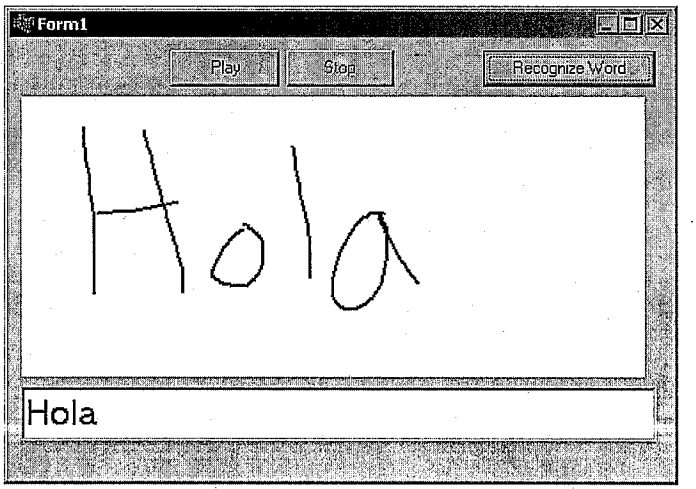
Ahora solo falta invocar a este adaptador desde la Interfaz de usuario (ver figura).



En esta interfaz el usuario dibuja la palabra con ayuda del Mouse y al presionar el boton "Recognize Word" se debe ejecutar el siguiente código:

```
void __fastcall TForm1::btnRecognizeWordClick(TObject *Sender)
{
   Adapters::RecognizerAdapter adapter(
      "http://localhost:1459/RecognizerService/Service.asmx?WSDL");
   HandWriting::Idiom idiom;
   HandWriting::RecognizedWord recognizedWord;
   recognizedWord = adapter.RecognizeWord(word, idiom);

   lstWords->Items->Add(recognizedWord.Word );

}
```

Como vemos en el código se especifica la dirección del servicio Web y le pasamos los parámetros necesarios para ejecutar la operación. El resultado se verá en la interfaz de la siguiente forma:



Como vemos la aplicación cliente ha invocado el servicio Web para que este reconozca la palabra dibujada.

De esta manera termínanos la implementación de la arquitectura de la solución propuesta.

6. Conclusiones

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

Hoy en día SOA esta siendo promovida en la industria como la siguiente generación de arquitecturas de software para ayudar a las organizaciones de tecnologías de información a cumplir sus retos más complejos.

7. Glosario

CORBA	Common Object Request Broker Architecture.
DCOM	Distributed Component Object Model.
Orquestación	Composición de servicios Web para soportar procesos de negocio definidos.
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
W3C	World Wide Web Consortium

8. Referencias

1	Web services ORG, http://www.webservices.org
2	SBS SYS. http://sys.sbs.de/
3	Servicios Web y Seguridad. http://sys,sbs.de/e/security/index.htm#web
4	Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications. http://www.ibm.com/developerworks/rational/library/510.html
5	W3C Working Group Note, Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
6	Patterns: Service-Oriented Architecture and Web Services. Mark Endrel, Jenny Ang. http://publibb.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg246303.htm
7	Composición semántica de los servicios Web. Grupo de ingeniería telemática, Universidad de Cauca Colombia. Noviembre 2004.
8	Enterprise SOA: Service Oriented Architecture Best Practices. By Dirk Krafzig, Karl Banke, Dirk Slama. Pretince Hall PTR.
9	Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. By Thomas Erl. Pretince Hall PTR.
10	Understanding SOA with Web Services. By Eric Newcomer, Greg Lomow. Addison Wesley Professional
11	Web Services and Service-Oriented Architecture: The Savvy Manager's Guide. by Douglas K. Barry
12	On the Road to SOA. Building Real-World Enterprise Webservices. By Charles Beckham, John Crupi, Todd Fast. Sun Microsystems.
13	The SOA implementation framework. By Jason Bloomberg.