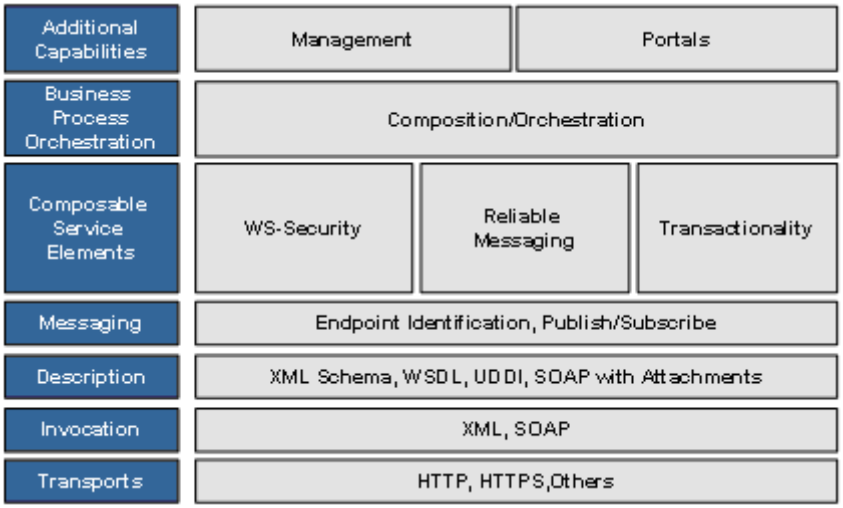

Figura 9. Pila de estándares de los servicios Web en los que trabaja WS-I

Aunque la propuesta de protocolos para servicios Web ha sido un área de trabajo muy activa y con un crecimiento muy rápido, su transición a estándares abiertos es inevitablemente mucho más lenta. Existen muy pocos protocolos que hayan finalizado su proceso de estandarización de forma adecuada. En [CBDI, 2006a] se puede encontrar el estado actual de adopción de los protocolos presentados.

3. ARQUITECTURA ORIENTADA A SERVICIOS (SOA)

Los sistemas de información son cada vez más complejos y se necesitan sistemas más exigentes que requieren cooperación; y cada vez son menos comunes las soluciones centralizadas y aparecen nuevos métodos de interacción (comunicaciones). El objetivo de las organizaciones es reducir los costes y maximizar la utilización de la tecnología existente; al mismo tiempo las empresas intentan ser más competitivas y avanzar en sus prioridades estratégicas.

Existen dos líneas para conseguir estos propósitos: La heterogeneidad y el cambio. Muchas empresas tienen diferentes sistemas, aplicaciones y arquitecturas de diferentes tecnologías y algunas bastante antiguas. Integrar productos de diferentes proveedores y de diferentes tecnologías es una ardua tarea, por eso cada vez más las aplicaciones están orientadas al servicio. Esto se puede ver en los gráficos del informe del CBDI Forum [CBDI, 2006b], que se muestra en la figura 10.

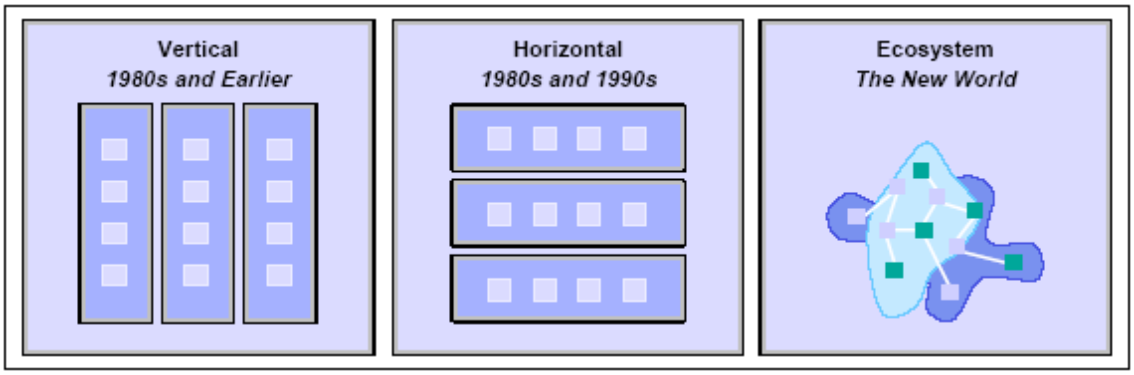

Figura 10. Evolución de los procesos de negocio

Por otra parte, en la figura 11 se puede apreciar la evolución hacia las arquitecturas orientadas a servicio.

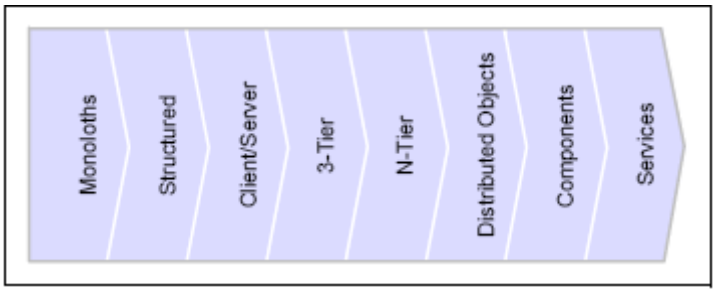

Figura 11. Evolución de las arquitecturas de sistemas software

Durante años la industria de la tecnología ha estado luchando para solucionar problemas como la heterogeneidad, la interoperabilidad (para que la solución escogida o la aplicación construida sea independiente de la plataforma y tecnología utilizada), incluso en la forma de recoger los requisitos o que las aplicaciones sean independientes del protocolo de comunicación. Han ido apareciendo distintas formas de plantear la construcción de aplicaciones, sin lugar a duda una de las más importantes para resolver los problemas anteriores a sido el enfoque de la orientación a objetos, aplicado tanto al análisis, diseño, como programación de aplicaciones, planteando el desarrollo de software como "un problema de dominio y una solución lógica desde la perspectiva de los objetos (cosas, conceptos o entidades)" [Larman, 2001]. Jacobson et al. [1992] definen estos objetos como "características con un número de operaciones y un estado que recuerda los efectos de las operaciones sobre ellos".

En el análisis orientado a objetos, tales objetos son identificados y descritos en el dominio del problema; y en el diseño orientado a objetos son transformados en objetos de software que serán implementados en un lenguaje orientado a objetos. De esta forma se reduce el esfuerzo de análisis y diseño de escenarios complejos y se facilita la reutilización.

A partir de la orientación a objetos han ido apareciendo otros "paradigmas" como la orientación a aspectos [POA, 2006], o la orientación a servicios; un servicio es generalmente implementado como una entidad software que puede ser accedida como una única instancia e interactúa con otras aplicaciones y otros servicios a través de un modelo de comunicación basado en mensajes. Antes de proceder a la descripción del concepto de arquitectura orientada a servicio, conviene recordar la terminología relacionada con los servicios Web:

- **Servicios:** Entidades lógicas con una funcionalidad definida y las reglas establecidas por una o más interfaces que son publicadas.

- **Proveedor de servicio (*Service provider*):** La entidad de software que implementa la especificación del servicio.

- **Consumidor del servicio (*Service consumer (o requestor)*):** La entidad de software que llama al proveedor de servicio. Tradicionalmente llamado "cliente". Un consumidor de servicio puede ser una aplicación u otro servicio.

- **Servicio localizador (*Service locator*):** Un tipo específico de proveedor de servicio que actúa como un registro y permite la búsqueda de interfaces y localizaciones de servicios.

- **Servicio intermediario (*Service broker*):** Un tipo específico de proveedor de servicio que puede pasar las peticiones de un servicio a uno o más proveedores de servicios adicionales.

3.1 CONCEPTO DE ARQUITECTURA ORIENTADA A SERVICIOS

Existen muchas definiciones de arquitectura, y también existen muchas interpretaciones de estas definiciones. La arquitectura de un sistema, en general, describe su estructura, a través de los siguientes aspectos:

- Los componentes que intervienen como bloques básicos del sistema.

- Conectores que describen los mecanismos de comunicación con otros sistemas y los mecanismos de interconexión entre los propios componentes de la arquitectura.

- Los flujos que muestran como una aplicación utiliza los componentes y los conectores para llevar a cabo su objetivo.

Se utiliza el acrónimo SOA para hacer referencia a la arquitectura orientada a servicio, el término SOA son las siglas inglesas de *Service Oriented Architecture*.

La Arquitectura Orientada a Servicio presenta una ventaja para la construcción de sistemas distribuidos, ya que contempla, la funcionalidad de las aplicaciones como servicios que pueden utilizar otras aplicaciones y otros servicios.

Una Arquitectura Orientada a Servicio está compuesta por elementos funcionales y elementos relacionados con la calidad de servicio, como se puede ver en la figura 12 y que se describen a continuación.

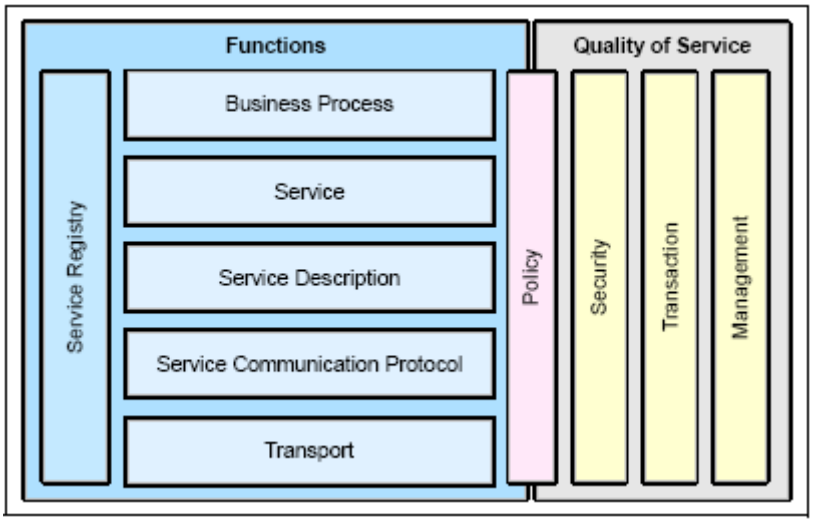

Figura 12. Elementos de una Arquitectura Orientada a Servicio

Aspectos funcionales:

- **Transporte:** Es el mecanismo usado para llevar peticiones de servicios desde el consumidor al proveedor del servicio, y las respuestas desde el proveedor del servicio al consumidor del servicio.

- **Protocolo de comunicación del servicio:** Es el mecanismo de comunicación establecido entre el proveedor del servicio y el consumidor del servicio.

- **Descripción del servicio:** Es el esquema establecido para describir qué es el servicio, cómo debe invocarse y que datos son requeridos para la invocación.

- **Servicio:** Describe un servicio que está disponible para utilizarse.

- **Proceso de negocio:** Es una colección de servicios, invocados de una manera particular, en una determinada secuencia y con unas reglas particulares para

llevar a cabo la funcionalidad de negocio requerida. Un proceso de negocio puede estar compuesto por servicios de diferente naturaleza e incluso en distintas localizaciones.

- **Registro de Servicio:** Es el repositorio de servicios y las descripciones que son usados por los proveedores de servicio para publicarlos, y para que los consumidores del servicio puedan invocarlos. El registro del servicio puede aportar la funcionalidad a los servicios que necesiten un repositorio centralizado.

Aspectos de la calidad del servicio:

- **Política:** Es un conjunto de condiciones o reglas sobre las cuales un proveedor del servicio hace un servicio disponible a los consumidores.

- **Seguridad:** Es el conjunto de reglas que pueden ser aplicadas para la identificación, autorización y el control de acceso a los consumidores de servicios.

- **Transacción:** Es el conjunto de atributos que pueden ser aplicados a un grupo de servicios para conseguir un resultado consistente. Por ejemplo si un grupo de tres servicios tienen que terminar para completar la función, todos tienen que estar completados y haber terminado su ejecución.

- **Gestión:** Es el conjunto de atributos que pueden ser aplicados para manejar a los proveedores del servicio o a los consumidores.

3.2 COLABORACIÓN EN UNA ARQUITECTURA ORIENTADA A SERVICIOS

En la figura 13 se pueden ver los elementos de colaboración existentes en una Arquitectura Orientada a Servicio:

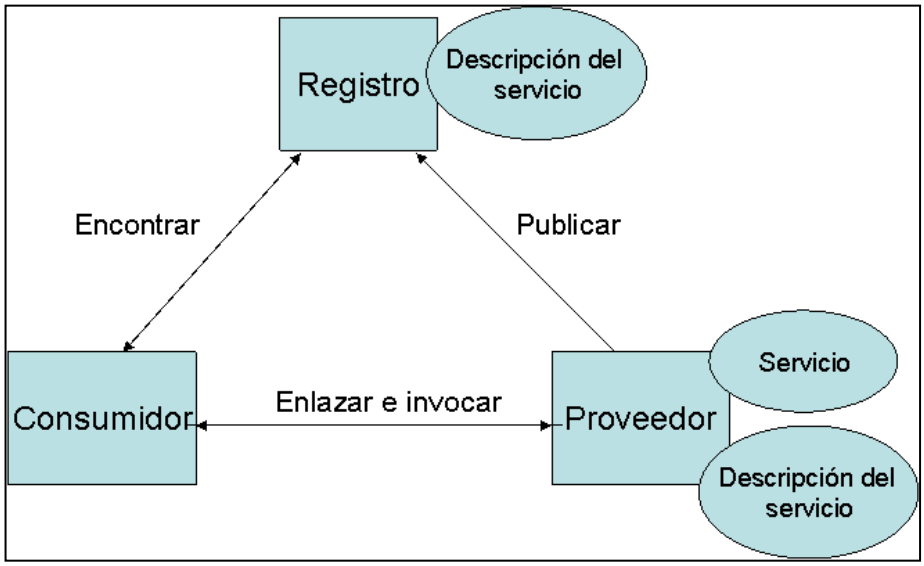

Figura 13. Elementos de colaboración en una Arquitectura SOA

Los roles de una Arquitectura Orientada a Servicio son:

- **Consumidor de servicio:** El consumidor de servicio es una aplicación, un módulo de software u otro servicio que requiere un servicio. Inicia la búsqueda en el registro de servicio, enlaza con el servicio a través del transporte y ejecuta la función del servicio de acuerdo con las reglas establecidas.

- **Proveedor de servicios:** El proveedor de servicios es una entidad que se puede acceder a través de la red y que acepta y ejecuta peticiones de los consumidores. Publica las interfaces de los servicios en el registro de servicios para que los consumidores puedan descubrirlos y puedan acceder a ellos.

- **Registro de servicios:** Un registro de servicios es el que permite que los servicios puedan ser descubiertos. Contiene un repositorio con los registros que están disponibles y permite la búsqueda de los proveedores de los servicios a través de las interfaces que han sido establecidas y que son de interés para los consumidores.

Las operaciones dentro de una arquitectura orientada a servicios son:

- **Publicación:** Para que los servicios puedan estar accesibles, un servicio tiene que tener una descripción, que debe ser publicada para que pueda ser descubierta e invocada por un consumidor de servicio.

- **Localización:** Un consumidor del servicio puede localizar un servicio realizando una búsqueda sobre el registro de servicios que cumpla algún criterio.

- **Enlazar e invocar:** Después de recoger la descripción del servicio, el consumidor del servicio puede invocar el servicio de acuerdo con la información de la propia descripción del servicio.

En la figura 14 se puede ver más claramente el entorno y forma de colaboración:

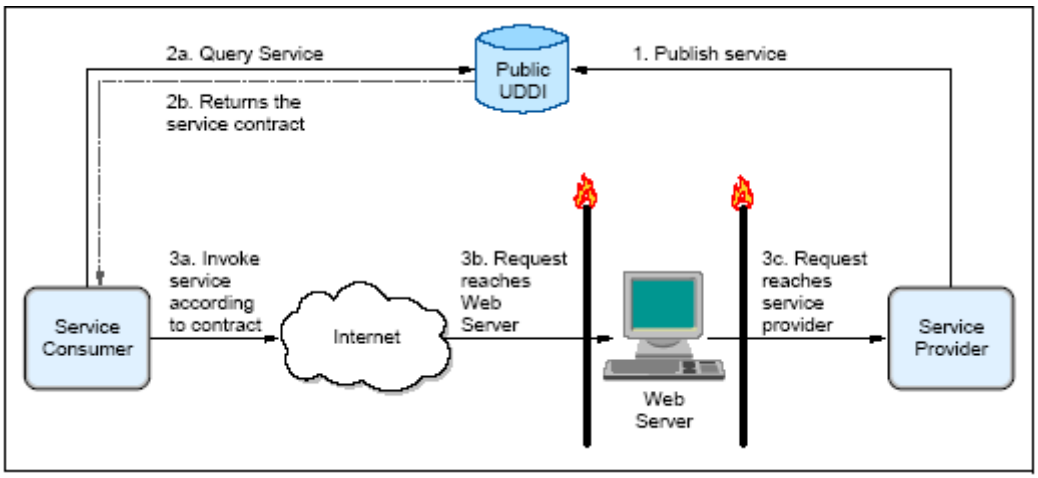

Figura 14. Colaboración en una Arquitectura SOA

Cómo ya se comentó en apartados anteriores, para poder realizar estas operaciones, existen distintas tecnologías; así, para realizar la descripción del servicio que especifica la forma de acceder al consumidor y poder interactuar con el proveedor se utiliza WSDL (*Web Services Description Lenguaje*) [W3C, 2001a], esta especificación establece el qué, el dónde y el cómo se accede a un determinado servicio. Y para poder hacer la localización, la integración y poder enlazar e invocar a un Servicio Web se utiliza UDDI (*Universal Description, Discovery and Integration*) [UDDI, 2007].

3.3 CARACTERÍSTICAS DE UNA ARQUITECTURA ORIENTADA A SERVICIOS

Para que el funcionamiento de una arquitectura orientada a servicio sea dinámica, tiene que cumplir las siguientes características [McGovern et al., 2001]:

- Los servicios tienen que ser modulares.
- Los servicios tienen que soportar la interoperabilidad.
- Los servicios tienen que tener la descripción perfectamente establecida.
- Los servicios tienen que ser transparentes a la localización.
- Los servicios tienen que ser independientes del lenguaje de implementación.
- Los servicios tienen que ser transparentes al protocolo de comunicación.
- Los servicios tienen que ser independientes del Sistema Operativo y del hardware utilizado.

3.4 BENEFICIOS DE UNA ARQUITECTURA SOA

Como ya se ha indicado, los procesos de negocio de una organización se van creando teniendo en cuenta la habilidad para cambiar rápidamente, la heterogeneidad y sobre todo la necesidad de reducción de costes. Para recordar la competitividad existente entre las distintas organizaciones, deben poder crear procesos de negocio y ofrecer servicios a sus clientes de forma que se puedan adaptar rápidamente a factores internos como adquisiciones, fusiones y reestructuraciones, o factores externos como son los requerimientos del mercado y de los clientes. Para las compañías es necesario llevar un equilibrio entre el coste, la efectividad, la calidad de sus procesos y la flexibilidad de la infraestructura tecnológica de que disponen.

Con una arquitectura orientada a servicio (SOA) se pueden alcanzar algunos beneficios para ayudar a las distintas organizaciones a conseguir unos procesos de negocios exitosos y dinámicos:

- **Solventar los problemas existentes:** Utilizando una arquitectura SOA se establece una capa de abstracción que permite a una organización continuar ofreciendo una innovación en tecnología, encapsulando los problemas en servicios que ofrecen como funciones de negocio. Las organizaciones pueden

continuar obteniendo valor utilizando recursos externos a través de los servicios en lugar de invertir y reconstruir las soluciones existentes.

- **Fácil de integrar y gestionar la complejidad:** El punto de integración en una arquitectura SOA es la especificación del servicio y no la implementación. De esta forma aporta la transparencia de la implementación y minimiza el impacto cuando la infraestructura o la implementación existente cambian. Mediante la especificación del servicio con las funciones implementadas o los problemas que resuelve, incluso en distintos sistemas, la integración resulta sencilla.

- **Acelerar la puesta en producción de los sistemas (*time-to-market*):** La habilidad de componer nuevos servicios haciendo uso de los ya existentes aporta una ventaja significativa a una compañía que necesita responder de una forma rápida a la demanda de negocio que le requieren. El ciclo de desarrollo se puede llegar a reducir, aunque también es importante saber establecer bien una base de requisitos funcionales y de pruebas.

- **Reducir costes y aumentar la reutilización:** Realizando una base de servicios que se pueden exponer y se pueden reutilizar, incluso si los servicios los ofrecen otras compañías, resultan mucho menos costosos los desarrollos, se necesita menos tiempo de codificación, se aumenta la reutilización en los desarrollos, existe menos duplicación de recursos, y por lo tanto, se aumenta el potencial y se reducen los costes.

- **Estar preparados para el cambio:** La arquitectura SOA permite a las organizaciones estar preparadas para el futuro. Los procesos de negocio se pueden crear mucho más fácilmente si se piensan y se crean en base a servicios gestionados y se dejan las interfaces de interconexión totalmente definidas, permitirá a las compañías seguir creciendo sin necesidad de invertir de nuevo en los mismos desarrollos, y además permite a los sistemas existentes seguir creciendo.

Con esto se puede comprobar que utilizando las arquitecturas SOA se aporta gran valor, pero también hay que recordar que migrar a una arquitectura orientada a servicio no es una tarea trivial. Más que una migración de toda una organización a una arquitectura SOA, se recomienda migrar el conjunto de funciones o artefactos que necesitan interoperar con otras organizaciones, y empezar a utilizar SOA para futuros desarrollos.

También se pueden ver los beneficios de una Arquitectura SOA desde el punto de vista empresarial; algunas de ellas son las siguientes:

- **Eficiencia:** Transforma los procesos de negocio en servicios compartidos con un menor coste de mantenimiento, mayor ROI (*Return Of Investment*).

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

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

3.5 SERVICIOS WEB Y ARQUITECTURA ORIENTADA A SERVICIOS

En apartados anteriores se presentó con detalle lo que eran los servicios Web y las tecnologías que les ofrecen soporte; en este apartado se justifica porque los servicios Web pueden llegar a ser una parte fundamental de una Arquitectura Orientada a Servicio.

Dentro de un proceso de negocio puede ser necesario utilizar funcionalidades de distintos sistemas y distintas localizaciones; para completar estas funcionalidades se utilizan los Servicios Web, que también tienen que ser identificados durante el proceso de análisis de una arquitectura orientada a servicio. Cada servicio tiene que estar bien definido mediante una interfaz (WSDL) para que pueda ser publicado, localizado e invocado. Dependiendo del proceso de negocio, el servicio puede ser publicado para que otras empresas puedan utilizarlo, o internamente para ser utilizado en los procesos de negocio internos de una determinada compañía.

Los servicios Web son una tecnología altamente adaptable a las necesidades de implementación de una Arquitectura Orientada a Servicio. En esencia, los servicios Web son la implementación de una especificación bien definida de una funcionalidad, es decir, son aplicaciones modulares que aportan una lógica del proceso de negocio como servicio que puede ser publicado, localizado e invocado en Internet [IBM, 2006]. Basados en los estándares XML [W3C, 1998], los servicios Web pueden ser desarrollados usando cualquier lenguaje de programación, cualquier protocolo y cualquier plataforma. Los Servicios Web pueden ser localizados y utilizados en cualquier momento, desde cualquier localización y usando cualquier protocolo y plataforma.

Pero es importante remarcar que los servicios Web no son la única tecnología que es usada para implementar una Arquitectura Orientada a Servicio. Existen ejemplos de organizaciones que utilizan con éxito una Arquitectura Orientada a Servicios donde utilizan, además de los servicios Web otras tecnologías de intercambio de mensajes y de

acceso a funciones remotas haciendo uso del estándar XML, como puede ser el protocolo XML RPC [Winer, 1999], que funciona exactamente igual que el protocolo RPC (*Remote Procedure Call*), a través de un túnel HTTP.

3.6 PATRONES DE DISEÑO DE UNA ARQUITECTURA ORIENTADA A SERVICIOS

También para la construcción de aplicaciones orientadas a servicio utilizando arquitecturas SOA, se han desarrollado una serie de patrones de diseño de software [GoF, 2003]. Con la utilización de patrones, que se pueden utilizar con cualquier metodología, la Arquitectura Orientada a Servicio resultante contendrá un nivel de abstracción mayor, asegurándose una mejor consistencia y rendimiento ya se han utilizado buenas prácticas de diseño.

Los patrones de diseño para construir una arquitectura orientada a servicio se pueden dividir en cinco categorías [Monday, 2003]:

1. **Aprendizaje:** Sirven para entender el entorno de los servicios Web. Dentro de esta categoría podemos encontrar:
   - *Service-Oriented Architecture*: Es el patrón que forma la arquitectura de los servicios Web como ya hemos visto anteriormente.
   - *Architecture Adapter*: Se puede ver como un patrón genérico que facilita la comunicación entre arquitecturas.
   - *Service Directory*: Este patrón facilita la transparencia en la localización de servicios, permitiendo realizar robustas interfaces para encontrar el servicio que realmente se quiere.

2. **Adaptación:** Estos patrones son los llamados básicos para conocer el funcionamiento del entorno de los servicios Web. En esta categoría nos encontramos:
   - *Bussines Object*: Un *business object* engloba a un concepto de negocio del mundo real como puede ser un cliente, una compañía o un producto, y lo que pretende este patrón es trasladar el concepto de objeto de negocio dentro del paradigma de los servicios Web.
   - *Business Process*: Este patrón se utiliza para tratar con procesos de negocio. En este momento existen dos especificaciones:
     - *Business Process Execution Lenguaje* (BPEL) propuesto por Bea Systems, IBM y Microsoft.
     - *Business Process Modeling Languaje* (BPML) propuesto por el resto de compañías que no están en el grupo anterior como pueden ser WebMethods, SeeBeyond, etc.

- *Bussines Object Collection*: Con este patrón se pueden realizar composiciones de procesos de negocio.
- *Asynchronous Business Process*: Este patrón es la evolución del patrón anterior *Bussines Process*.

3. **Cambios:** Aunque los servicios Web permiten llamadas asíncronas, la implementación del servicio puede estar basado en paso de mensajes; también son importantes los servicios basados en eventos, estos patrones se basan en patrones tradicionales como el *Observer* o el patrón Publicación/Suscripción. En esta categoría podemos encontrar:
   - *Event Monitor*: Es un patrón para crear formas efectivas para integrar aplicaciones sin la intervención de otros componentes. El escenario más común donde se utiliza este patrón es para aplicaciones EAI (*Enterprise Application Integration*) [EAI, 2006].
   - *Observer Services*: Este patrón representa la manera más natural de detectar cambios y actuar en consecuencia.
   - *Publish/Subscribe Services*: Es la evolución del patrón *Observer*, mientras que el patrón *Observer* se base en el registro, el patrón Publicación/Suscripción se base en notificaciones, esto permite que distintos servicios puedan enviar la misma notificación.

4. **Redefinición:** Estos patrones permiten acceder al comportamiento de un servicio que está implementado en un lenguaje. Ayudan a entender el entorno del servicio Web y a moldear este entorno de acuerdo con nuestras necesidades. En esta categoría podemos encontrar:
   - *Physical Tires*: Este patrón ayuda a estructurar mejor la lógica de negocio de los servicios Web, e incluso se puede utilizar para controlar le flujo de negociaciones que puede llegar a producirse utilizando el patrón Publicación/Suscripción.
   - *Connector*: Este patrón se suele utilizar con el anterior para resolver los posibles problemas que surgen en la suscripción.
   - *Faux Implementation*: Es una alternativa para resolver los problemas que surgen en la utilización de eventos en los servicios Web. Es simplemente un "*socket abierto*" que recibe conexiones y aporta las respuestas para los distintos eventos.

5. **Flexibilidad:** Para crear servicios más flexibles y optimizados. En esta categoría se encuentran:
   - *Service Factory*: Es uno de los patrones más importantes y permite la selección de servicios y aporta flexibilidad en la instanciación de los componentes que crean los servicios Web. Este patrón también se suele utilizar con el patrón *Service Cache* para aportar una mayor flexibilidad

en el mantenimiento de las aplicaciones que utilizan servicios Web, aportando un mayor ROI a las aplicaciones.

- *Data Transfer Object*: Este patrón aporta rendimiento, ya que permite recoger múltiples datos y enviarlos en una única llamada, reduciendo el número de conexiones que el cliente tiene que hacer al servidor.
- *Data Transfer Collection*: Este es una extensión del anterior, ya que el patrón *Data Transfer Object* se puede aplicar a una colección de objetos de negocio. Este objeto puede devolver un grupo de atributos comunes de una colección de objetos.
- *Partial Population*: Este patrón permite a los clientes seleccionar únicamente los datos que son necesarios para sus necesidades y sólo recuperar del servidor lo necesario. Este patrón además de rendimiento aporta mayor ancho de banda en la red.

Algunos patrones utilizan otros; por ejemplo, el patrón *Business Process* usa los patrones *Business Object* y el *Business Object Collection*. Y el *Service-Oriented Architecture* hace uso de los patrones *Service Directory* y *Architecture Adapter*.

3.7 BUSINESS PROCESS EXECUTION LANGUAGE (BPEL)

La implementación ideal de un servicio exige resolver algunos inconvenientes técnicos inherentes a su modelo:

- Los tiempos de llamada no son despreciables, gracias a la comunicación de la red, tamaño de los mensajes, etc. Esto necesariamente implica la utilización de mensajería fiable.

- La respuesta del servicio se ve afectada directamente por aspectos externos, como problemas en la red, configuración, etc. Estos deben ser tenidos en cuenta en el diseño, desarrollándose los mecanismos de contingencia que eviten la parálisis de las aplicaciones y servicios que dependen de él.

- Debe manejar comunicaciones no fiables, mensajes impredecibles, reintentos, mensajes fuera de secuencia, etc.

Según lo anterior, se puede imaginar que la construcción de un servicio es una tarea mucho más complicada que la de un simple componente distribuido. Por esto, el servicio debe publicar una interfaz (por ejemplo, utilizando WSDL) fácilmente localizable en la red. Esta interfaz debe servir como un contrato de servicio, donde se describen cada una de las funciones que provee, e incluso los niveles de prestación de

servicio (SLA: *Service Level Agreement*). Esta interfaz debe estar claramente documentada de manera que sea muy fácil implementar una conexión.

Cuando se usan múltiples servicios para implementar un sistema, es muy fácil que la comunicación entre estos sea difícil de controlar. Por ejemplo, se puede tener un servicio que llama a otros servicios, algunos de los cuales llama a otros servicios, y de esta manera, muy fácilmente el sistema se vuelve inmanejable y un sistema grande puede terminar con múltiples dependencias. Detectar un problema de rendimiento o funcionalidad se puede volver muy complicado.

La forma tradicional para crear de manera rápida y sencilla nuevas aplicaciones que utilicen las ya existentes son los sistemas de gestión de flujos de trabajo (*workflows*).

Un flujo de trabajo especifica aspectos tales como:

- La secuencia de acciones a realizar por cada entidad.
- Los datos intercambiados entre las "entidades (aplicaciones, servicios)" y la manera en que deben ser transformados.
- Reglas para la toma de decisiones.
- Restricciones a satisfacer.

Algunas ventajas que ofrecen los sistemas de flujos de trabajo son las siguientes:

- Crear un nuevo proceso de negocio no implica programar (al menos, en gran parte).
- El sistema gestor del *workflow* se encarga de algunas tareas complejas como la transaccionalidad o la mensajería.
- La respuesta a cambios es más rápida.
- No es necesario escribir cada vez código hecho a medida para las aplicaciones (esto no siempre es cierto pero, al menos, una vez añadida una funcionalidad, queda disponible para todos los procesos de negocio).
- Facilita la implantación de políticas globales: autenticación, seguridad, etc.

La nueva evolución de los *workflows* se basa en la orquestación de los servicios Web: conectar servicios entre sí para obtener procesos de negocio de alto nivel.

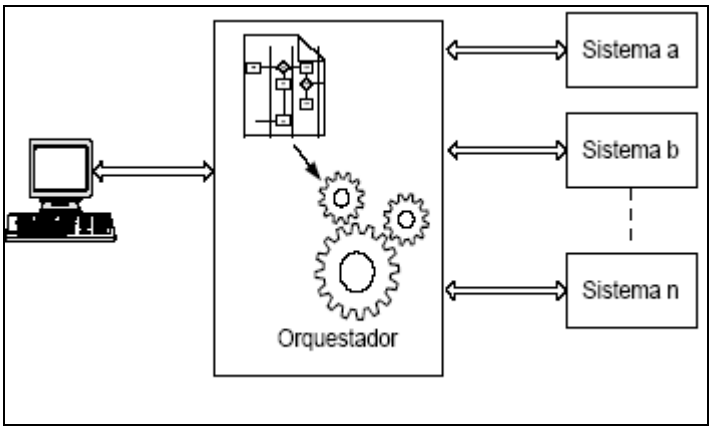

Figura 15. Orquestación de Servicios Web

Orquestación es un término relativamente nuevo definido por analogía a la ejecución de una pieza musical por parte de una orquesta. Una orquesta no es un conjunto de músicos tocando cada uno su propio instrumento al mismo tiempo, sino que es mucho más que eso, una orquesta en el acto de ejecutar una obra, es un conjunto de instrumentos que bajo una bien definida coreografía, la interpretan juntos de acuerdo a su arreglo musical. Cada músico debe entrar y salir a su tiempo ejecutando la parte que le corresponde en ese momento, según el director de la orquesta de acuerdo a la coreografía de la obra.

De la misma manera, un proceso de negocio debe ser orquestado bajo su propia coreografía y mediante un orquestador dirigir la ejecución de los componentes que actúan en conjunto para que, ejecutando cada cual lo que le corresponde en el momento indicado, compongan la ejecución de ese proceso de negocio (figura 15).

La orquestación proporciona secuencia y sincronización: el "qué" ejecutar y el "cuando" hacerlo. El orquestador dirige la ejecución de una cierta coreografía para articular un proceso de negocio.

3.7.1 Orquestación versus Coreografía

Como hemos visto los servicios Web se pueden combinar de dos maneras:

- Orquestación
- Coreografía

En la orquestación, que suele ser usada para procesos de negocios privados o para procesos centrales, donde un proceso coordina un grupo de diferentes operaciones, los servicios involucrados no "conocen" (y no necesitan conocer) todo el proceso del que forman parte. Sólo el coordinador central de la orquestación conoce el objetivo global;

de esta forma, la orquestación es centralizada con definiciones explícitas de las operaciones y con el orden de invocación de los servicios Web (figura 16).

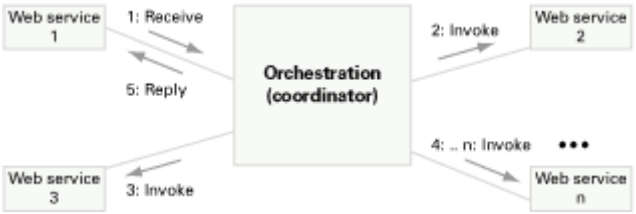

Figura 16. Composición de Servicios Web con orquestación

En la coreografía no existe un coordinador central, cada servicio Web involucrado en la coreografía conoce exactamente cuando se tiene que ejecutar y las operaciones y con quien tiene que interactuar. La coreografía se utiliza en los procesos de negocio públicos con intercambios de mensajes. Todos los servicios Web necesitan conocer quien es el siguiente servicio que se va a ejecutar, qué operación, que mensaje se intercambia y el tiempo de ejecución (figura 17).

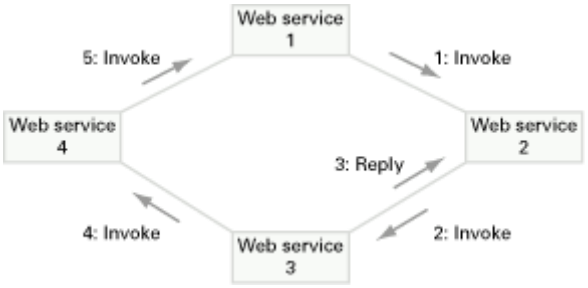

Figura 17. Composición de Servicios Web con coreografía

Desde la perspectiva de la composición de los servicios Web para la ejecución de procesos de negocio, la orquestación es más flexible y tiene las siguientes ventajas sobre la coreografía:

- La coordinación es centralizada conducida por un coordinador conocido.

- Los servicios Web pueden ser incorporados sin afectar al proceso al que se incorporan, y no tienen porque conocer cual es el objetivo del proceso al que se están incorporando.

Para especificar la composición de servicios Web se puede utilizar el lenguaje BPEL [2007], que soporta dos formas de describir los procesos de negocio que soportan la orquestación y la coreografía:

- **Procesos ejecutables:** Permiten especificar exactamente los detalles de los procesos de negocio. Estos procesos cumplen con el paradigma de la orquestación.

- **Protocolos abstractos de negocio:** Permiten la especificación del intercambio de mensajes públicos únicamente entre participantes. No incluyen detalles internos de los flujos de los procesos y no son ejecutables. Siguen el paradigma de la coreografía.

Después de las distintas normas propuestas parece que la norma BPEL4WS (Business Process Execution Language for Web Services) [IBM, 2005] va a ser el estándar de la orquestación de los servicios Web. Es utilizado para procesos abstractos (que definen conversaciones y protocolos para el uso de un servicio Web o de la forma de colaborar de varios servicios Web) y para procesos ejecutables (que son flujos de trabajo dónde cada entidad involucrada es un servicio Web y que, a su vez, ofrecen al exterior una interfaz de servicio Web).

Normalmente un proceso BPEL4WS se compone de:

- Un fichero con el proceso a ejecutar (.BPEL)

- Una serie de ficheros WSDL de apoyo (definiciones).

Las actividades en BPEL pueden verse como nodos en un grafo, dónde los enlaces entre nodos determinan el flujo de control entre actividades. Las actividades tienen dos elementos opcionales para establecer dependencias entre los enlaces del grafo que conforman el flujo:

- **Target.** La actividad es destino del enlace especificado. Esto quiere decir que la actividad no comenzará hasta que dicho enlace se active (normalmente, al finalizar otra actividad).

- **Source.** La actividad es fuente del enlace especificado. Cuando la actividad termine, activará el enlace, posiblemente desencadenando la ejecución de otras actividades que tengan ese enlace como target.

Manejando estas dependencias, es posible construir estructuras condicionales, flujos en paralelo, etc. En definitiva, lo que se trata es de fijar unas directivas para que puedan cooperar todos los procesos empresariales, y así organizados mediante la utilización de distintos flujos de trabajo, se consigue el objetivo que se pretende.

El lenguaje BPEL está basado en el esquema XML [W3C, 2004c], en el protocolo SOAP (*Simple Object Access Protocol*) [W3C, 2003], y el lenguaje de descripción de servicios Web WSDL (*Web Services Descripton Language*) [W3C, 2001a] y permite lo siguiente:

- Enviar mensajes XML y recibir mensajes síncrona y asíncronamente desde servicios Web remotos.

- Manipular estructuras de datos XML

- Manejar eventos y excepciones

- Diseñar flujos de procesos

- Deshacer la continuidad del proceso en un flujo definido cuando ocurre una excepción

3.7.2 Diseño de un proceso de negocio

Un proceso especifica el orden exacto en el cual los servicios Web participan y deben ser invocados secuencialmente o en paralelo. También se pueden expresar comportamientos condicionales. Por ejemplo, una invocación de un servicio Web puede depender del valor de una invocación anterior. Se pueden construir bucles, declarar variables o copiar y asignar valores. Como los procesos se pueden representar gráficamente, ya que son esencialmente grafos de actividades, se pueden expresar usando diagramas de actividad UML [Jacobson et al., 2000].

En un escenario típico, los procesos BPEL reciben una petición, y para completarla, el proceso invoca a los servicios Web que participan en el proceso y después responde al que ha hecho la petición original. En todo el ciclo del proceso, como existe comunicación entre distintos servicios Web, hay que tener muy bien definida la descripción WSDL.

A continuación se va a presentar un ejemplo de un proceso BPEL: un proceso BPEL consiste en “pasos” y a cada paso se denomina “actividad”; una actividad se basa en primitivas que representan los constructores básicos y son usadas para tareas comunes como las siguientes:

- Invocar a otros servicios Web, usando la etiqueta <*invoke*>.

- Esperando a que el cliente invoque al proceso enviando un mensaje, usando <*receive*> (recibiendo una petición).

- Generando una respuesta para las operaciones síncronas, usando <*reply*>.

- Manipulando variables, usando <*assign*>.

- Indicando errores y excepciones, usando <*throw*>.

- Esperando por algún espacio de tiempo, usando <*wait*>.

- Terminando el proceso, usando <*terminate*>.

Se pueden combinar estas y otras actividades primitivas para definir algoritmos complejos que especifiquen exactamente los pasos de los procesos de negocio. Para combinar estas actividades primitivas BPEL soporta algunas estructuras, las más importantes son:

- Secuencia <*secuence*>, permite la definición de un conjunto de actividades que serán invocadas en una secuencia ordenada.

- Flujo <*flow*>, para definir un conjunto de actividades que serán invocadas en paralelo.

- Estructura <*switch*>, para implementar caminos alternativos.

- Estructura <*while*>, para definir bucles.

- La habilidad de seleccionar uno o más caminos alternativos, usando <*pick*>.

En un proceso BPEL también se pueden utilizar enlaces, usando <*partnerLink*>, y declarar variables utilizando <*variable*>.

Para entender como funciona un proceso de negocio con BPEL, se va a describir un proceso de negocio para los empleados de una agencia de viajes [Oracle, 2005]: El cliente invoca al proceso especificando el nombre del empleado, el destino, la fecha de salida y la fecha de llegada. El proceso BPEL primero comprueba el estado del empleado, asumiendo que el servicio Web existe a través del cual puede comprobarse esta situación. Después el proceso BPEL comprobará el precio para el billete de vuelo en dos compañías (American Airlines y Delta Airlines), de nuevo se asume que ambas compañías de vuelo tienen servicios Web para poder hacer esta operación. Finalmente, el proceso BPEL seleccionará el precio más bajo y retornará el plan de vuelo al cliente.

Cuando se define un proceso de negocio en BPEL, esencialmente se está definiendo un nuevo servicio Web que es una composición de servicios existentes. La interfaz de la nueva composición BPEL utiliza un conjunto de tipos de puertos a través de los cuales provee operaciones como cualquier otro servicio Web. Para invocar el proceso de

negocio descrito en BPEL, se tiene que invocar al servicio Web resultado de la composición. En la figura 18 se puede ver el esquema del proceso.

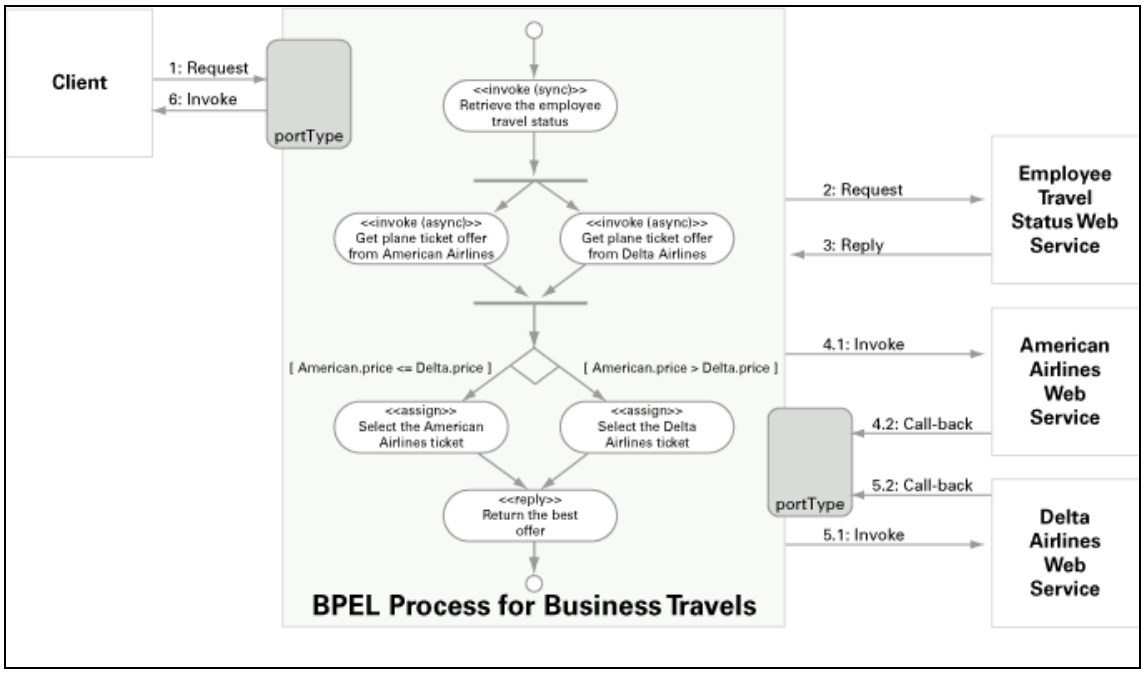

Figura 18. Ejemplo de un proceso BPEL

3.7.3 Arquitectura de un servidor BPEL

Para que se puedan ejecutar procesos de negocio BPEL se necesita un entorno de ejecución para que se puedan ejecutar los procesos; actualmente en el mercado existen distintos servidores que se integran con los servidores de aplicaciones, permitiendo desplegar, monitorizar y gestionar los distintos procesos.

En la figura 19 se puede ver la arquitectura del servidor BPEL de Oracle, (*Oracle BPEL Process Manager*) [Oracle, 2005]. Está desarrollado en Java y es compatible con cualquier servidor de aplicaciones compatible J2EE como por ejemplo Oracle Application Server OC4J, JBoss o Bea Weblogic.