Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección de Computación

Generación dinámica de workflows organizacionales escritos en BPEL4WS

Tesis que presenta

Lic. César Sandoval Hernández

para obtener el Grado de

Maestro en Ciencias en la Especialidad de

Ingeniería Eléctrica

Director de la Tesis

Dr. José Oscar Olmedo Aguirre

México, D. F. Noviembre 2004

Resumen

Los mercados se están abriendo al uso del Internet como una forma eficaz de conducir prácticas comerciales. Esto ha provocado el surgimiento de un nuevo paradigma computacional llamado Servicios Web que tiene como principal aplicación integrar los procesos organizacionales internos y externos de las empresas. Esta integración requiere coordinar secuencias de actividades comerciales y los lenguajes composicionales son los más adecuados para lograr esta coordinación. BPEL4WS, un lenguaje de modelación de procesos de negocio que permite coordinar invocaciones de servicios Web elementales, se está convirtiendo en un estándar. Sin embargo, en BPEL4WS no es posible expresar una definición genérica de workflows, con la cual se puedan describir formas cada vez más complejas de colaboración entre negocios.

Esta tesis propone una infraestructura que permite la generación dinámica de procesos de negocio a partir de workflows genéricos (plantillas). Se ha construido un Administrador de Procesos de Nagocio (APN) capaz de recuperar, de un repositorio de plantillas BPEL4WS, la plantilla adecuada para un proceso comercial específico. El APN concretiza, ejecuta e invoca dinámicamente el proceso de negocio, produciendo en tiempo de ejecución procesos ad hoc a las necesidades del cliente.

Se han construido procesos comerciales genéricos que representan la compra de un producto al menor precio, menor tiempo de entrega, etc. lo cual demuestra la utilidad de nuestra infraestructura. Haciendo uso de este trabajo junto con una arquitectura de intermediación, pequeñas organizaciones pueden automatizar muchos de sus procesos comerciales sin necesidad de hacer inversiones importantes en desarrollo de software.

Abstract

In recent years, Internet has been broadly used by enterprises as a way to do commercial transactions. In response, a new computational paradigm called Web services has emerged to integrate inter- and intra- organizational processes. This integration requires coordinating sequences of commercial activities; and this coordination is reached using compositional languages. BPEL4WS is becoming the de facto business process modeling language that allows enterprises to coordinate elementary Web services. Nonetheless, a major problem in BPEL4WS modeling is that the language does not support generic definitions of workflows with which we can describe increasingly complex forms of business collaboration.

In this thesis we propose an infrastructure that can be used to generate dynamic business processes instantiating generic workflows (templates). A Business Process Manager (BPM) has been developed to retrieve from a BPEL4WS template repository, the necessary template to execute a specific commercial process. The BPM instantiates, executes and invokes dynamically the business process, producing at execution time ad hoc processes.

We have created generic business processes that represent the purchase of a product at the lowest price, minimum delivery time, etc. that demonstrate the capabilities of our infrastructure. By using a brokering architecture, small organizations can automate many of their business processes without making large investments in software development and deployment.

Agradecimientos

Al CONACYT por el apoyo económico que me permitio concluir la maestría.

A la Fundación Telmex cuyo apoyo ha sido integral en todos los aspectos de mi vida.

Al dr. José Oscar Olmedo Aguirre por su asesoría que sin duda ha sido invaluable para la terminación de esta tesis.

A Giner Alor Hernández por sus consejos y ayuda en la elaboración de esta tesis.

Dedicatorias

A mis amados padres *Angela Hernández Lindero* y *Pascual Sandoval Peralta* quienes han sido el motor de mi vida. Y de quienes me siento tremendamente orgulloso por su tenacidad ante ella.

A mis hermanos *Martha Sandoval Hernández* y *Ulises Sandoval Hernández* quienes en todo momento me apoyaron y motivaron para concluir mi tesis. Y quienes son parte de lo más valioso que la vida me ha dado.

A mi amada puchupú con quien he compartido tantos hermosos momentos y me ha enseñado maravillosas lecciones de vida que me han dado la fortaleza de continuar con mis objetivos.

A *Dios*, divino ser quien me sigue apoyando en los pasos que doy en mi vida.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. La Web semántica

La World Wide Web ha sido desde su origen hasta nuestros días un medio eficaz para la presentación de información. Inicialmente la Web había sido concebida sólo con el propósito de intercambiar información de cualquier índole entre un grupo relativamente aislado de investigadores. Pero a medida que fue transcurriendo el tiempo, comenzó a tener una mayor aceptación en todo el mundo, a grado tal que no sólo ha revolucionado la comunicación entre personas sino que también ha revolucionado nuestra sociedad, cambiando significativamente nuestro estilo de vida. Sin embargo, la creciente complejidad de los mercados digitales demanda el uso de tecnología más sofisticada que permita la generación de aplicaciones con mayores alcances.

Actualmente la información en la Web es comprensible únicamente por humanos, siendo prácticamente imposible para las computadoras interpretar la información que se presenta [1]. La Web semántica se considera en la actualidad como el siguiente nivel en la

evolución de la Web. En ella se visualiza la posibilidad de dar significado al contenido de las páginas Web y de hacer inferencias sobre la información contenida en dichas páginas. Esto permitirá la creación de aplicaciones que realicen tareas cada vez más complejas y que ayuden en la difícil tarea de hacer más fácil la vida de los seres humanos. En síntesis, la Web semántica se puede ver como el siguiente paso de la Web en la cual la información está bien definida y tiene un sentido para las computadoras lo que les permite trabajar en un ambiente cooperativo.

Para incorporar métodos de razonamiento automatizado sobre la información, la Web Semántica necesita un lenguaje que permita expresar la información y establecer reglas sobre ésta. Existen dos tecnologías que están encaminadas a resolver este problema: el Lenguaje de Marcado Extensible (XML) [2] y el Ambiente de Descripción de Recursos (RDF) [3]. La primera básicamente describe la información intercambiada entre las aplicaciones pero no le agrega ninguna semántica, es decir, no dice nada acerca del significado de la información. Mientras que la segunda sí permite agregar significado a la información la cual es estructurada en forma de tripletas (sujeto, verbo y predicado). A través de RDF se pueden hacer aseveraciones de que un objeto en la Web posee un valor. Además, el Universal Resource Identifier (URI) provee un medio para definir conceptos universales los cuales se definen en algún lugar en la Web. La Web semántica también requiere del uso de ontologías que permitan hacer comparaciones y combinaciones de información a partir de las cuales se puedan hacer inferencias. Esta tecnología permitirá dotar a la Web de la capacidad de recuperar información más precisa que aquella que se puede recuperar a través de los buscadores actuales.

Otro aspecto fundamental que plantea la Web semántica es la automatización de tareas. Se pretende crear aplicaciones con base en sistemas multi - agentes que sean capaces de recolectar automáticamente el contenido de las páginas Web localizadas en distintos sitios en la red, procesar la información e intercambiarla con otros agentes. Un agente

es un programa que realiza determinadas tareas para su usuario. A diferencia de otros programas, los agentes tienen las propiedades generales de:

- confiabilidad: el agente realiza su función de la manera señalada por el usuario, esto es, la probabilidad de que el agente realice su función prevista sin incidentes por un período de tiempo especificado y bajo condiciones indicadas debe ser alta.
- personalización: el agente debe aprender a hacer tareas exclusivas para un usuario y
- autonomía: el agente debe ser capaz de realizar acciones en representación del usuario.

Aunque en la actualidad no se han explotado al máximo las características de los agentes, se cuenta con la tecnología necesaria para la automatización de servicios basados en la Web.

El presente trabajo hace contribuciones específicamente en la automatización de tareas planteada por la Web semántica. Dicha automatización ha sido lograda en gran medida gracias a los avances que se han conseguido en la computación distribuida de la cual se hablará en la siguiente sección.

1.2. La computación distribuida

La computación distribuida es un área de investigación dentro de las Ciencias de la Computación que tiene como finalidad distribuir componentes de una aplicación en diferentes computadoras conectadas a una red de tal forma que cada uno de ellos realice una tarea

específica. Uno de los requerimientos más importantes de la computación distribuida es el uso de estándares que especifiquen cómo dichos componentes se comunican entre ellos. Se puede decir que los más importantes han sido el Distributed Component Object Model (DCOM) [4] y el Common Object Request Broker Architecture (CORBA) [5]. El primero es una propuesta de Microsoft y el segundo del Object Management Group (OMG). Tanto DCOM como CORBA permiten la invocación remota de métodos de componentes que se encuentran distribuidos en la red.

Sin embargo, ambas tecnologías tienen limitaciones que han impedido la expansión de su uso en las aplicaciones orientadas a Internet. Una de ellas es que dichas tecnologías se caracterizan por ser excesivamente complejas tanto en la programación como en la instalación de la infraestructura que las soporta. Además, no permiten el envío de mensajes asíncronos, lo que limita significativamente el desarrollo de las aplicaciones. Otra limitante relacionada con estas dos tecnologías y que pudiera ser la más importante es la dificultad que se tiene para desarrollar aplicaciones que sean interoperables. Los fabricantes de software han hecho sus propias implementaciones, impidiendo que las diferentes aplicaciones basadas en CORBA o DCOM operen libremente unas con otras. Más aún, DCOM es enteramente dependiente de la plataforma, lo que lo confina a ejecutarse solamente en plataformas Windows.

Por otro lado, la forma que se tenía de hacer llamadas a procedimientos remotos utilizando CORBA era a través del Remote Procedure Call (RPC). Una implementación orientada a objetos es el Remote Method Invocation (RMI) el cual es una propuesta enteramente basada en Java y por esta razón independiente de la plataforma. Aunque su uso se ha visto restringido por los esquemas de seguridad manejados en las intranets ya que muchas empresas utilizan *firewalls* para evitar ataques del exterior a su red interna. Generalmente las políticas de seguridad consisten en cerrar todos los puertos excepto los usados por HTTP o FTP. Sin embargo, las aplicaciones construidas sobre esta tecnología

usan muy diversos puertos que no necesariamente corresponden a los de HTTP o FTP, lo que generalmente les impide trabajar correctamente. Los servicios Web han resultado ser una buena opción para resolver esta problemática. En la siguiente sección se habla más detalladamente de ellos.

1.3. El paradigma de los servicios Web

El uso de los servicios Web en el World Wide Web se extiende rápidamente debido a la creciente necesidad de comunicación e interoperabilidad entre aplicaciones. Los servicios Web proveen esencialmente un medio estándar de comunicación entre diferentes aplicaciones de software. Están basados en estándares abiertos, lo que permite a las aplicaciones comunicarse más libremente e independientemente de las plataformas en que se están ejecutando. Además de que se encuentran encima de una infraestructura ampliamente aceptada y fácil de usar, de alguna manera se puede decir que el intercambio de la información se hace usando lenguajes estandarizados, mientras que su manipulación depende del lenguaje de programación que se está utilizando.

Los protocolos de comunicación que los servicios Web utilizan se basan en XML el cual, como su nombre lo indica, es un lenguaje de marcado extensible que permite describir cualquier tipo de información. En los servicios Web se hace uso de lenguajes derivados de XML que ayudan a describir las operaciones que se van a ejecutar o la información a intercambiar con otro servicio.

El mayor logro de los servicios Web es que han permitido a las aplicaciones comunicarse entre ellas de manera independiente del lenguaje de programación y de la plataforma, es decir, consiguen mayor interoperabilidad. Existe una gran desarrollo tecnológico relacionado con los servicios Web donde destaca el directorio Universal Description, Discovery

and Integration (UDDI) [6], el Simple Object Access Protocol (SOAP) [7] y el Web Services Description Language (WSDL) [9]. UDDI define un registro y protocolos asociados para la búsqueda y localización de servicios Web. Aunque como se ha dicho anteriormente, debido a que XML no agrega semántica a la información, dichas búsquedas son limitadas. SOAP permite codificar documentos XML para que puedan ser transmitidos en la red usando protocolos de transporte como HTTP. En tanto que WSDL define una interfaz que describe una colección de operaciones que pueden ser accedidas remotamente.

También se ha creado una infraestructura de coordinación que permite la orquestación de tareas delegadas a servicios Web. Esta orquestación de servicios Web se logra gracias a los lenguajes composicionales que permiten la automatización de tareas propuesta por la Web semántica y de la cual hablaremos en la siguiente sección.

1.4. La orquestación de servicios Web

Los servicios Web por sí mismos proveen una forma eficaz de integrar aplicaciones de negocio orientadas a la Internet. Sin embargo, debido a que un negocio es en esencia la suma de sus procesos [10] el verdadero valor de los servicios Web radica en la conexión de sus servicios, lo que provee un mayor valor a la organización [11].

Como se explica en [11] existen dos perspectivas desde las cuales se puede abordar el problema de conectar un conjunto de servicios Web para que colaboren entre sí. Estas perspectivas son la orquestación y la coreografía. La orquestación de servicios Web consiste en proveer los medios necesarios para establecer un proceso de negocio ejecutable que permita la interacción entre servicios Web internos o externos. El proceso de negocio es dirigido por una de las partes del negocio. Mientras que la coreografía es más distribuida en el sentido de que ninguna de las partes controla el proceso del negocio. De hecho,

en la coreografía, cada parte involucrada en el proceso describe el rol que juega en la interacción. Se puede decir que en la orquestación se plantea la ejecución de un proceso desde un punto de vista único y general, en tanto que en la coreografía se deja a cada una de las partes del proceso ejecutarse desde su perspectiva sin que ninguna de ellas controle la conversación.

En la actualidad existen principalmente dos lenguajes estándares para realizar orquestación y coreografía. El primero es el Web Service Choreography Interface (WSCI) el cual describe desde la perspectiva de cada uno de los servicios Web su participación en el intercambio de los mensajes, es decir, más apegado a la coreografía. Mientras que el segundo es el Business Process Execution Language for Web Services (BPEL4WS) [12] el cual se apega tanto a la coreografía como a la orquestación.

BPEL4WS define una notación para especificar el comportamiento de un proceso de negocio basado en los Servicios Web, es decir, modela un *workflow*. Un *workflow* consiste en la automatización de procesos de negocio, parcial o total, durante la cual los documentos, información o tareas se transfieren de un participante a otros, de acuerdo a un conjunto de reglas de procedimiento [13]. BPEL4WS provee también un modelo basado en una alta interoperabilidad que facilita la integración de procesos automatizados en los espacios internos de una corporación y entre las empresas.

En el presente trabajo se usan ampliamente las propiedades de orquestación de BPEL4WS. Se hace especial énfasis en las limitaciones que presenta este lenguaje las cuales serán abordadas en la siguiente sección.

1.5. Planteamiento del problema

Cuando un cliente hace una petición a alguna empresa que ofrece sus servicios en Internet, ésta realiza una serie de procesos internos y/o externos para satisfacer dicha petición. Estos procesos que pueden resultar simples o complejos, pueden ser modelados y ejecutados haciendo uso de lenguajes composicionales como el Business Process Execution Language for Web Services (BPEL4WS).

Para ilustrar lo anterior se puede pensar en el problema de un viajero que desea volar de México a Madrid, a las 12:00 p.m. el próximo viernes. Este viajero desea buscar y comprar sus boletos de avión por Internet. Para lograr su propósito necesita acceder a las páginas Web de las diferentes aerolíneas y buscar cuál de todas ofrece vuelos el día y hora que necesita. También podríamos pensar que le gustaría rentar un automóvil que podría utilizar durante su estancia en España y hacer las reservaciones en un hotel madrileño. En la Figura 1.1 se puede ver el proceso que debe seguir el viajero en cuestión.

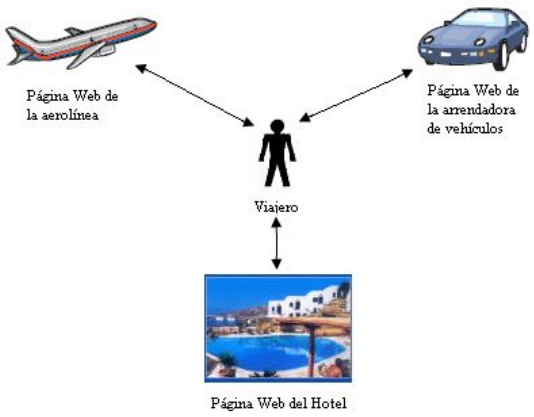

Figura 1.1: Pasos efectuados por el viajero para reservar su vuelo y habitación en Madrid; y rentar su automóvil.

Lo que parecía ser una tarea relativamente simple del viajero se ha convertido en una pérdida de tiempo durante la búsqueda y ha desembocado en una serie de molestias para él. Para lograr la automatización de todos estos pasos se puede utilizar precisamente BPEL4WS. Al hacer uso de este lenguaje el cliente únicamente tendría que hacer su petición, lo que se traduciría en una serie de invocaciones paralelas a los servicios Web de las aerolíneas, hoteles y arrendadoras de vehículos. En este contexto, el proceso de reservación puede ser visto como un proceso de negocio; mientras que las aerolíneas, arrendadoras y hoteles fungen como los socios comerciales del proceso de negocio. Una vez completadas las peticiones, el sistema que invocó los servicios Web procedería a determinar la mejor opción dado un criterio de búsqueda específico como el menor precio posible por ejemplo. Lo que normalmente llevaría más de 30 minutos a una persona efectuar todos esos pasos (habiendo consultado varias aerolíneas, hoteles y arrendadoras de vehículos), podría reducirse a no más de un minuto. En la Figura 1.2 se puede ver esquemáticamente el proceso que ha de ejecutarse para satisfacer la petición del viajero.

Sin embargo, uno de los mayores problemas de BPEL4WS que limita seriamente su funcionamiento es que sólo describe instancias de procesos de negocio y no procesos genéricos de negocio. Las instancias de procesos de negocio consisten en un conjunto de actividades determinadas en tiempo de diseño por algún experto en la generación de *workflows*. Este *workflow* está dirigido a la solución de un problema específico. Mientras que los procesos genéricos de negocio ofrecen una solución más general para un problema específico. En ellos se abstraen las actividades recurrentes de un proceso de negocio y en tiempo de ejecución se completan para resolver un problema en particular. En este sentido, se puede ver a un proceso de negocio genérico como una lista de actividades definidas en tiempo de diseño a las que sólo les hace falta incluir cuáles y cuántos servicios Web serán agregados en tiempo de ejecución. Supongamos el caso en el que un cliente desea hacer la compra de sus productos por Internet, en donde, como suele suceder, existe más de una tienda que satisface sus necesidades. El cliente en cuestión puede desear hacer su compra distribui-

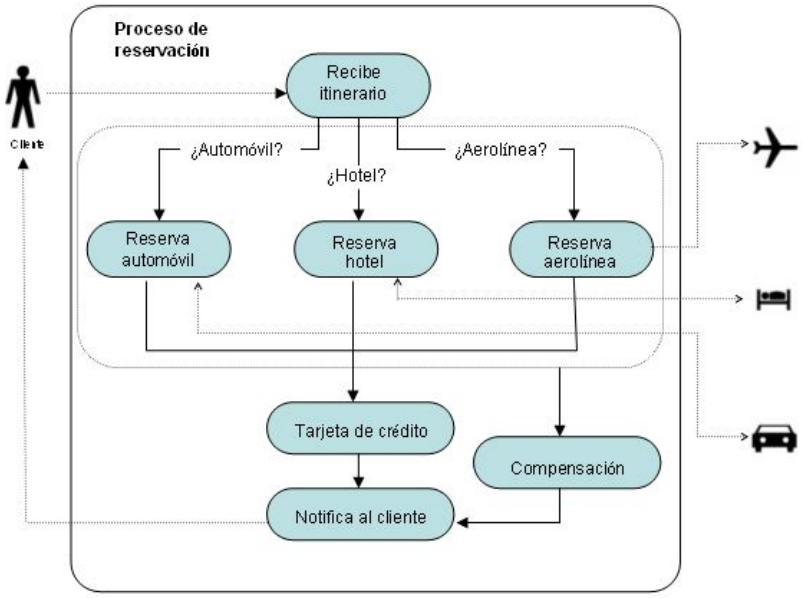

Figura 1.2: Proceso de reservación que se ejecuta automáticamente para satisfacer la petición del cliente.

damente en un conjunto de tiendas. El actual esquema bajo el cual trabaja BPEL4WS impide hacerlo debido a que se necesita agregar dichas tiendas al *workflow* en tiempo de diseño, lo cual es imposible ya que dicha información sólo es descubierta en tiempo de ejecución.

1.6. Propuesta de solución

Como solución se planea generar una serie de plantillas escritas en BPEL4WS de propósito genérico, las cuales serán depositadas en un repositorio. Estas plantillas consisten en un conjunto de instrucciones invariables que representan las partes de un proceso que no cambian para problemas de la misma clase. Así pues, cuando un cliente requiera hacer búsquedas, compras o algún otro tipo de procesos por Internet, pueda acceder al repo-

sitorio públicamente disponible; y concretizar y ejecutar los procesos ya automatizados que son de su interés. El proceso de concretización consiste básicamente en completar las plantillas con la información proveniente de los socios comerciales.

Mientras que el proceso de ejecución tiene que ver con el envío en tiempo de ejecución de la plantilla concretizada a la máquina virtual de BPEL4WS. Esto resulta, por otro lado, realmente benéfico porque se podría automatizar una gran cantidad de procesos genéricos y de amplia utilización en Internet.

Es necesario, por tanto, desarrollar una infraestructura de coordinación que permita la descripción y ejecución de procesos de negocio, la integración de un número arbitrario de procesos de negocio y la traducción de consultas simples de alto nivel a un conjunto de actividades básicas junto con un programa de ejecución.

Esta infraestructura está orientada en principio a facilitar la integración de sistemas de búsqueda y compra de servicios y productos. Como se ha visto, actualmente los procesos de búsqueda y compra en Internet no han sido automatizados al máximo. Los usuarios necesitan realizar una gran cantidad de pasos para poder realizar la búsqueda y compra de un producto que satisfaga sus necesidades. El uso de tecnología más sofisticada que ayude a los usuarios a obtener los resultados que buscan en menos tiempo y con menos esfuerzo, es imprescindible. La infraestructura que se propone ayuda a alcanzar dicho objetivo.

1.7. Objetivo General

El objetivo general de este trabajo de tesis consiste en diseñar y construir un administrador de procesos de negocio con base en BPEL4WS. Para ello se necesita desarrollar una infraestructura de coordinación en Internet que permita la descripción y ejecución de

procesos de negocio, la integración de un número arbitrario de procesos de negocio y la traducción de consultas simples de alto nivel a un conjunto de actividades básicas junto con un programa de ejecución.

1.8. Objetivos Particulares

1. Diseñar y construir el administrador de procesos de negocio partiendo de la máquina virtual BPEL4WS.
2. Diseñar y construir un portal para acceder a los servicios del administrador de procesos de negocio.
3. Diseñar procesos de negocio genéricos escritos en BPEL4WS.
4. Traducir peticiones de servicio (consultas) a la ejecución de programas BPEL4WS.

1.9. Metodología

Para lograr las metas arriba mencionadas, se seguirán los siguientes pasos.

1. Para registrar los procesos de negocio se hará:
   a) Un registro de servicios. Registrar servicios que ofrecen empresas en el servidor UDDI.
   b) Un registro de procesos de negocios. Registrar procesos genéricos de negocios en una base de documentos BPEL4WS.
2. Para usar los procesos de negocio se hará:

a) La identificación. Cuando se reciba una petición de servicio, se extrae la información relevante que permita identificar al servicio y a algunos de sus parámetros.

b) La traducción. Recuperar del registro de procesos el documento BPEL4WS indexado por el nombre del servicio y prepararlo para su ejecución.

c) La concretización. Obtener de un sistema de intermediación la información de la empresa (*accessPoint*, *overviewURL*, etc.) y sustituirla en los parámetros correspondientes del documento BPEL4WS.

d) La ejecución. Ejecutar el proceso de negocio descrito en el documento BPEL4WS completamente concretizado.

e) La respuesta. Enviar las respuestas de la ejecución al solicitante.

1.10. Contribuciones

Este trabajo presenta diferentes contribuciones en el área de la automatización de tareas. Dentro de estas contribuciones destacan:

1. La definición de *workflows* genéricos. Estos tienen como objetivo abstraer las tareas más usadas en Internet como la compra y la búsqueda de productos.

2. El diseño de un repositorio de plantillas BPEL4WS. Este repositorio es similar a UDDI y tiene el propósito de extender las capacidades de BPEL4WS de tal forma que se puedan recuperar plantillas de *workflows* que resuelven un problema y adecuarlas a nuestras necesidades.

3. Los procesos de concretización. Estos procesos permiten la creación de *workflows ad hoc* a las necesidades del cliente basándose en ontologías.

4. Un sistema capaz de administrar procesos de negocio con base en BPEL4WS.

5. Un portal que permita acceder a los servicios del sistema administrador de procesos de negocio.

1.11. Organización del documento

Esta tesis ha sido estructurada de la siguiente manera. El capítulo 2 muestra en detalle la tecnología relacionada con los servicios Web y los trabajos relacionados con la presente tesis. El propósito de este capítulo es mostrar un panorama amplio en el que está inmerso este trabajo. Por otro lado, en el capítulo 3 se plantea un caso de estudio y el funcionamiento general del Administrador de Procesos de Negocio (APN) sobre éste. Además se muestran las interfaces del portal con las que interactúa el usuario. En el capítulo 4 se presenta el diseño del APN a partir de las perspectivas organizacional, de datos y funcional. En el capítulo 5 se muestran los puntos técnicos de la implementación del APN. Y finalmente, en el capítulo 6 se exponen las conclusiones donde se muestran los resultados, contribuciones y trabajo a futuro.

Capítulo 2

Tecnología y trabajos relacionados

Este capítulo tiene dos objetivos principales. El primero es plantear los conceptos básicos relacionados con este trabajo que introduzcan al lector en el área. El segundo es mostrar la relevancia del trabajo realizado exponiendo los problemas que aún se tienen que resolver en el contexto presentado en la primera parte.

En la primera parte de este capítulo se muestra la tecnología relacionada con los servicios Web. Se presentan las capas de mensajería, descripción de servicios, publicación y descubrimiento y composición de los servicios Web. También se describen conceptos que prometen la solución de muchos de los problemas que han limitado los alcances de la Web.

En la segunda parte, se presentan los trabajos involucrados con esta tesis y los enfoques planteados para resolver el problema de agregar dinámicamente socios comerciales al proceso de negocio. Se analizan estos trabajos y se muestran sus ventajas y desventajas sobre nuestra propuesta.

2.1. Tecnología relacionada con los servicios Web

Los servicios Web son aplicaciones modulares que proveen una infraestructura sistemática y extensible para la interacción de aplicaciones. Esta infraestructura está dividida en tres áreas: los protocolos de comunicación, las descripciones de servicio y el descubrimiento de servicios [14]. Sin embargo, también se puede considerar el área de composición de servicios fundamental dentro del esquema de los servicios Web. En la Figura 2.1 se muestra una representación gráfica de la infraestructura de los servicios Web.

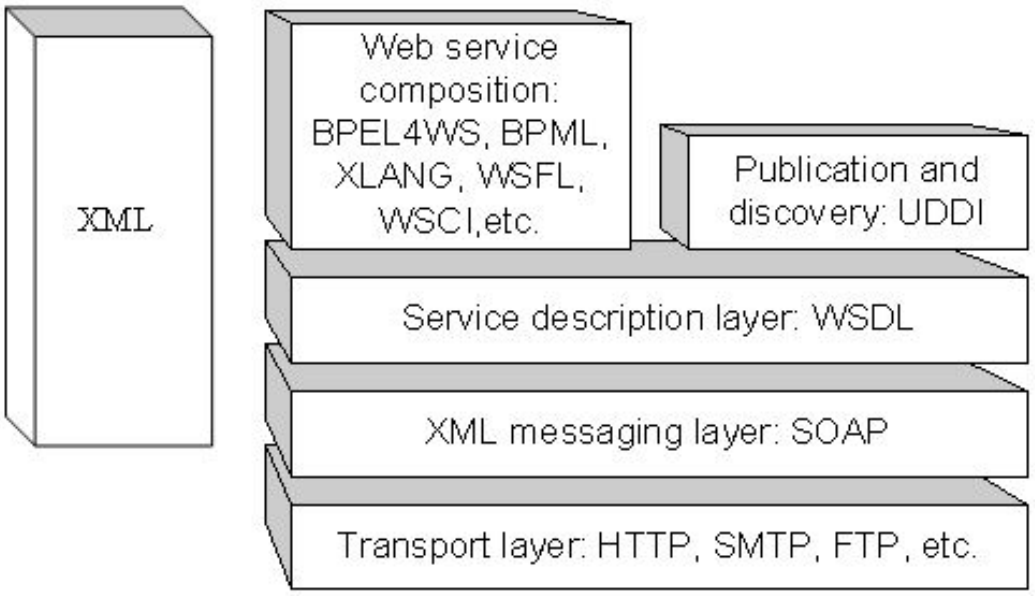

Figura 2.1: Estructura en capas de las tecnologías relacionadas con los servicios Web.

En la Figura 2.1 se observa que los servicios Web se encuentran encima de protocolos ampliamente usados en las aplicaciones Web y se basan en lenguajes derivados de XML. La capa de transporte es la misma que tradicionalmente se utiliza para el envío de información en la red (HTTP, SMTP, FTP). Arriba de esta capa se encuentra la capa de mensajes implementada por SOAP e inmediatamente arriba se encuentra la capa de descripción de

servicios. Finalmente, en la última capa se encuentran tanto los lenguajes composicionales como el registro UDDI. En esta sección se describe cada una de estas capas.

2.1.1. El lenguaje de marcado extensible XML

La red se caracteriza por ser un ambiente inherentemente heterogéneo. En ella reside una variedad enorme de aplicaciones pertenecientes a muy diversas organizaciones. De ahí la necesidad de mecanismos de comunicación que sean independientes de la plataforma y del lenguaje de programación. XML se ha consolidado en este contexto como el lenguaje estándar para la descripción de información que promueve la interoperabilidad. Básicamente permite la descripción de una gran cantidad de información lo que ayuda a los servicios Web a manejarla más fácil y libremente.

Más que un lenguaje, XML es un conjunto de reglas orientadas a la creación de lenguajes de marcado semánticamente ricos [15]. XML estructura la información a través de etiquetas llamadas elementos (*elements*). Un elemento es un contenedor de XML que consiste en una etiqueta de inicio, un contenido (caracteres, subelementos o ambos) y una etiqueta final. Un ejemplo de un elemento se muestra a continuación.

$< tesis\ fecha =$ "*septiembre de 2004*"$>$

$< autor >$

César Sandoval Hernández

$< /autor >$

$< title >$

Generación dinámica de workflows organizacionales escritos en BPEL4WS

$< /title >$

$< /tesis >$

Las etiquetas $< tesis >$ y $< /tesis >$ son las etiquetas de inicio y final respectivamente. Los elementos *autor* y *title* corresponden a los subelementos del elemento *tesis*. El atributo *fecha* especifica una característica del elemento *tesis*. En este ejemplo se puede apreciar la enorme facilidad con que se puede estructurar la información. De forma intuitiva se puede deducir, a partir del código anterior, que la información así presentada se refiere a una *tesis* cuyo *autor* es *César Sandoval Hernández* y *título* es *Generación dinámica de workflows organizacionales escritos en BPEL4WS*. Esta tesis tiene como característica que fue hecha en *septiembre de 2004* . Evidentemente también se pueden agregar otros elementos que caracterizan a una tesis como los *capítulos*, *secciones*, etc.

El elemento inicial en un documento XML (en este caso *tesis*) es conocido como el elemento raíz y los subelementos como nodos ya que la estructura de un documento XML tiene la forma de un árbol.

Por otro lado, un documento XML debe estar bien formado y ser válido. A grosso modo se puede decir que un documento XML está bien fomado si

- Existe un solo elemento raíz en todo el documento XML,
- Las etiquetas de inicio y final se anidan apropiadamente,
- Todas las entidades referenciadas directa o indirectamente desde el documento XML están bien formadas y
- Todas las restricciones de la especificación [2] se cumplen.

En la especificación [2] se puede encontrar una explicación más formal de un documento bien formado.

Mientras que un documento XML válido referencia y satisface un esquema. Un esquema es un documento que tiene como objetivo definir los elementos, atributos y estructura de la instancia de un documento XML que son legales. Más simplemente se puede decir que un esquema sirve para definir el vocabulario, número y lugar de los elementos y atributos que son válidos en el lenguaje de marcado que se está creando. Su uso puede ser el de validar las instancia de documentos XML para asegurar la precisión de los valores de un campo y la estructura de un documento. La precisión de los campos es revisada con respecto al tipo del campo (por ej. una cantidad que debe ser entero o el dinero que debe ser decimal). La estructura de un documento se revisa para que contenga los nombres de elementos y atributos válidos, el número de hijos correcto y los atributos requeridos.

El primer lenguaje de definición de esquemas fue el *Document Type Definition* (DTD). Sin embargo, presenta varias deficiencias que han hecho que se utilice cada vez menos. Las principales deficiencias son que no tiene una sintáxis XML, ni suficientes tipos de datos y no soporta los espacios de nombre. Aunque se han creado otros lenguajes para esquemas como el XML Schema que han superado por mucho estos problemas.

Un concepto importante dentro de XML, es el de los espacios de nombre los cuales son un mecanismo para crear nombres de elementos y atributos universalmente únicos. La importancia de esto radica en que con los espacios de nombre se evita la colisión entre elementos que tengan nombres idénticos y pertenezcan a diferentes lenguajes de marcado. También permite mezclar diferentes lenguajes de marcado sin ambigüedad. En XML los espacios de nombre están compuestos de un prefijo y una parte local. Un ejemplo de un elemento completamente calificado es:

$< xsd : integer >$

La parte local es *integer* y el prefijo es *xsd*. El prefijo es simplemente una abreviación del espacio de nombre real. El espacio de nombre real es un Identificador de Recursos Uniforme único (URI). Un ejemplo de una declaración real del espacio de nombre para el ejemplo anterior es:

$< xsd : schema\ xmlns : xsd =$ "*http://www.w3.org/2001/XMLSchema*"$>$

En este ejemplo se define que se está usando el elemento llamado *integer* correspondiente al espacio de nombres *http://www.w3.org/2001/XMLSchema*.

Ahora bien, para manipular y acceder a la información contenida en un documento XML se requiere de un *parser*. Un parser convierte los elementos de un documento XML en *tokens*. Existen dos formas de hacer un *parse* a un documento XML: usando el Simple API for XML (SAX) o a través del Document Object Model (DOM). SAX aplica un estilo basado en eventos donde cada información en el documento instancia genera un evento correspondiente en el *parser* cuando el documento es recorrido. SAX es usado para documentos XML muy grandes o en ambientes donde se carece de memoria. Por otro lado, DOM es en general el más usado para hacer un *parse* a un documento XML. DOM es básicamente un modelo de datos, usando objetos, para representar documentos XML. En este trabajo se hace uso extensivo de DOM para representar los elementos de un documento XML como objetos Java y así poder manipularlos.

Finalmente, XML provee un medio efectivo para manipular la información intercambiada entre los servicios Web a través de la especificación XPath. XPath es un lenguaje ampliamente aceptado que se usa para denotar y extraer fragmentos de documentos XML [16]. Sin embargo, en la mayoría de los casos, XPath no es suficientemente expresivo para manipular efectivamente esta información. BPEL4WS ha hecho extensiones consistentes de funciones, expresiones y consultas, para mejorar la expresividad de XPath. Estas extensiones han sido usadas extensivamente en este trabajo para recuperar la información

resultante de una petición.

Para ilustrar el uso de XPath, supongamos que una aplicación pide a otra el precio de un producto. En respuesta, el servicio Web envía información general acerca del producto donde se encuentra el precio. La información puede estar distribuida en la forma que se muestra en la Figura 2.2.

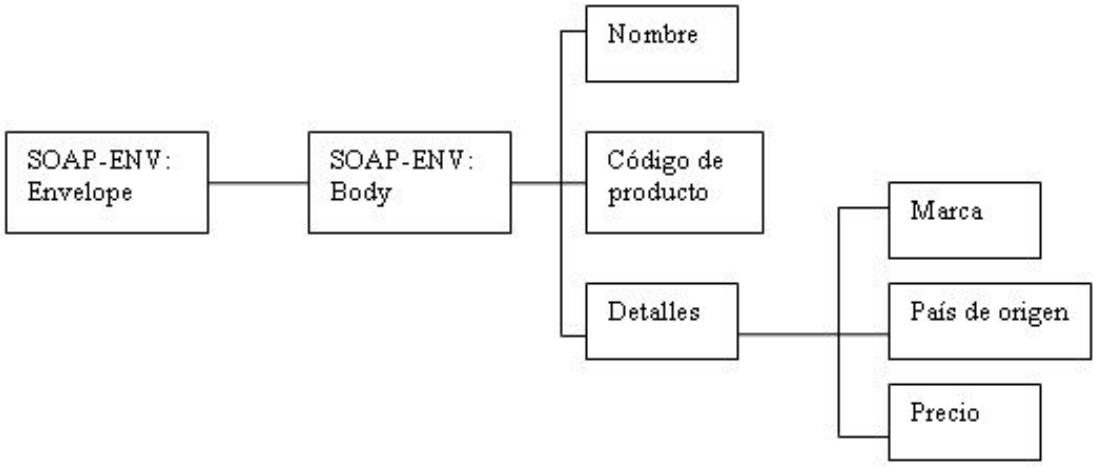

Figura 2.2: Ejemplo de una respuesta proveniente de un proveedor de servicio.

Como se puede apreciar en la Figura 2.2, se debe navegar por la estructura del mensaje de respuesta a través de información que puede ser irrelevante para las necesidades del cliente. En este ejemplo, se debe construir una ruta escrita en XPath, con la ayuda de funciones, expresiones y/o consultas, que describa la trayectoria que va desde el *Envelope* hasta el *Precio* para obtener el precio del producto.

En realidad la respuesta que se presenta en la Figura 2.2 tiene el formato de una respuesta SOAP. A continuación se explica la estructura de los mensajes SOAP.

2.1.2. La capa de mensajería SOAP

La capa de transporte usada por BPEL4WS se conoce como el Simple Object Access Protocol (SOAP) [17], el cual es un protocolo ligero usado para hacer envíos de mensajes y llamados a procedimientos remotos. Este protocolo ha sido impulsado por el World Wide Web Consortium (W3C) [8] para el intercambio de mensajes entre aplicaciones.

SOAP se sitúa arriba de una variedad de protocolos de transporte como HTTP o SMTP. BPEL4WS usa SOAP para intercambiar información estructurada con servicios externos a través de actividades de comunicación. La estructura de un mensaje SOAP es bastante simple aunque muy poderosa. Se divide en

- Un sobre. Este es el elemento raíz que identifica al documento XML como un mensaje SOAP.
- Opcionalmente una cabeza. Este elemento contiene información específica de la aplicación como información de autenticación, pago, etc. y se agrega al sobre.
- Un cuerpo. La información enviada a o recibida de un servicio Web está contenida en la parte del cuerpo. El cuerpo se agrega también dentro del sobre.

La estructura de un mensaje SOAP es como la que se muestra a continuación.

$<?xmlversion = 1{,}0? >$

$< soap : Envelope\ xmlns : soap = http : //www.w3.org/2001/12/soap - envelope >$

$< soap : Header >$

...

$< /soap : Header >$

$< soap : Body >$

...

$< store : GetPrice\ xmlns : store = http : //www.mycompany.com/prices >$

$< store : productCode > 431718 < /store : productCode >$

$< /store : GetPrice >$

$< /soap : Body >$

$< /soap : Envelope >$

Es necesario hacer notar que el espacio de nombres

$xmlns : soap = http : //www.w3.org/2001/12/soap - envelope >$

que se encuentra en el elemento em Envelope es fijo. No se puede cambiar porque esto generaría errores en el momento en que se envía el mensaje.

Como se puede ver en el ejemplo, el cuerpo del mensaje contiene la información que se desea enviar. En este caso, se podría enviar a una tienda el código de un producto cuya declaración se encuentra en el espacio de nombres $xmlns : store = http : //www.mycompany.com/prices >$ a través de la invocación de la operación $GetPrice$ de algún servicio Web. La respuesta que se espera es el precio de dicho producto también en forma de un mensaje SOAP.

2.1.3. El lenguaje de descripción WSDL

Aunque BPEL4WS usa diversas especificaciones, la más importante es el Web Services Description Language (WSDL) [9]. WSDL es un lenguaje basado en XML desarrollado por Microsoft e IBM para describir servicios Web como colecciones de puntos de comunicación donde los mensajes entre el proveedor de servicio y el cliente son intercambiados. En esencia, un documento WSDL describe una interfaz del servicio Web y provee la información de contacto para sus usuarios [14]. En un *workflow*, tanto el proceso como los servicios Web asociados a este proceso se modelan como servicios WSDL. Una estructura general de un documento WSDL se muestra en la Figura 2.3.

```
<definitions ...>
  <types>
       <!- En esta sección se define una
       lista de tipos de elementos -->
  </types>
  <message>
       <!- Definición de los mensajes
       intercambiados con servicios web
       externos -->
  </message>
  <portType>
       <!- Definición del portType accedido
       por el cliente -->
    <operation>
       <!- Definición de los mensajes de
       entrada y salida ofrecidos por el
       proceso -->
    </operation>
  </portType>
</definitions>
```

Figura 2.3: Estructura general de un documento WSDL.

En un documento WSDL se definen tipos de datos, mensajes, puertos, operaciones, socios, etc. que ayudan a establecer la comunicación entre un *workflow* y sus socios comerciales.

A continuación se describen cada uno de ellos.

Los tipos de datos definen los datos que se intercambian entre el proveedor de servicios y el solicitante. Estos tipos de datos pueden ser primitivos o complejos. Los primeros corresponden generalmente a cadenas, enteros o flotantes y se definen en un esquema estándar de la W3C [18]. Mientras que los últimos están compuestos por tipos primitivos o complejos.

Los mensajes están ligados a un protocolo de red concreto y consisten de una colección de tipos de datos. Un mensaje se puede ver como el conjunto de parámetros que se asocian a una operación de entrada o salida.

Las operaciones describen el intercambio de mensajes. Un proceso BPEL4WS representa todos sus socios e interacciones con estos socios en términos de las interfaces WSDL abstractas (*portTypes* y operaciones).

Finalmente, el *PartnerLinkType* sirve para establecer la relación en términos conversacionales entre dos servicios a través de la definición de los roles que cada servicio tiene en la conversación y de la especificación del *portType* expuesto por cada servicio para recibir mensajes.

2.1.4. El repositorio de servicios Web UDDI

UDDI [6] provee dos tipos de servicio: publicación y consulta. El primero define operaciones que permiten el registro, modificación y eliminación de negocios y servicios. Algunos ejemplos son $save\_business, save\_service, delete\_business$, $delete\_service$, por mencionar algunos. Mientras que el servicio de consulta define operaciones que permiten realizar búsquedas sobre negocios y servicios en un nodo UDDI como $find\_business, find -$

service, $find\_tModel, get\_businessDetail$, $get\_serviceDetail$, entre otros. Los negocios interesados en el registro de sus servicios Web en algún nodo UDDI deben usar el servicio de publicación. En la Figura 2.4 se muestra el esquema general usado por los servicios Web para publicar y registrar sus servicios Web.

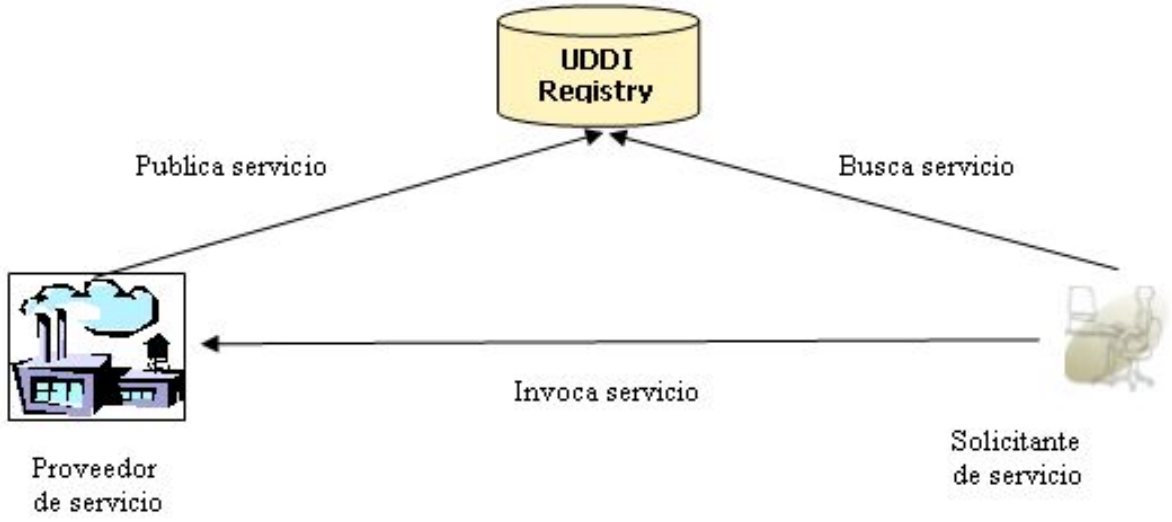

Figura 2.4: Estructura para el descubrimiento de servicios Web.

La información del negocio y sus servicios se almacena en una entidad llamada *BusinessEntity*. Esta entidad es un documento XML que según la especificación de UDDI es la más importante dentro de su estructura. Al igual que un directorio telefónico el elemento *BusinessEntity* divide la información en tres categorías: páginas blancas, páginas amarillas y páginas verdes. Las páginas blancas contienen información de contacto como nombre, teléfono, dirección o e-mail. Mientras que las páginas amarillas proveen información relacionada con la categoría del negocio basada en ontologías como las establecidas por el North American Industry Classification System (NAICS) [7] o el United Nations Standard Products and Services Code (UN/SPSC) [8], por mencionar algunas. Finalmente las páginas verdes ofrecen información de los servicios en la cual se especifica cómo otros negocios pueden integrarse.

2.1.5. El lenguaje composicional BPEL4WS

BPEL4WS permite la definición de procesos de negocio que usan servicios Web y de procesos de negocio que exponen su funcionalidad como servicios Web. Un proceso de negocio consiste en la especificación del orden de ejecución de operaciones de una colección de servicios Web, la información compartida entre estos servicios Web, los socios involucrados y cómo están involucrados en el proceso de negocio y la administración de las excepciones.

BPEL4WS define la funcionalidad de un proceso de negocio como servicio Web a través de lo que se conoce como *abstract businesses protocols* (protocolos de negocios abstractos). Un protocolo de negocio abstracto especifica únicamente los intercambios de mensajes públicos entre los participantes pero no su orden de ejecución. En contraste, un proceso ejecutable, modela el comportamiento de los participantes en una interacción de negocio específica, es decir, un *workflow* privado.

BPEL4WS se fundamenta en el Web Services for Business Process Design (XLANG) de Microsoft [19] y en el Web Services Flow Language (WSFL) de IBM [20]. De hecho, BPEL4WS combina los constructores de lenguaje estructurados en bloques de XLANG con una notación orientada a grafos que tiene sus raíces en WSDL.

BPEL4WS también está estrechamente relacionado con las especificaciones del $WS - Coordination$ y del $WS - Transaction$. La primera describe cómo los servicios Web pueden usar los contextos de coordinación predefinidos para que sean asociados a un rol particular en una actividad de colaboración [21]. Mientras que la última provee una infraestructura que dota a las actividades coordinadas de una semántica de transacción [22].

Un documento BPEL4WS está dividido en tres partes las cuales describen la información, las actividades de coordinación y las actividades de comunicación [23]. Las etiquetas de

información se usan para definir un conjunto de socios externos y el estado de un *workflow*. Las etiquetas que representan las actividades de coordinación definen el comportamiento del proceso a través de estructuras de flujo de control tradicionales. Finalmente, las etiquetas relacionadas con las actividades de comunicación y que también son parte de las estructuras del flujo de control definen la comunicación con otros servicios Web, a través de las actividades de coordinación enviando y recibiendo información. La estructura general de un documento BPEL4WS se muestra en la Figura 2.5.

Los *partnerLinks* permiten definir los servicios con los cuales el proceso de negocio interactúa. Un *partnerLink* especifica el *rol* del *PartnerLinkType* que el proceso acepta (*myRole*), y el rol que debe ser aceptado por el socio (*partnerRole*). Los servicios Web que espera el proceso y que son implementados por el socio son referenciados por *partnerRole* y los servicios Web provistos por el proceso y que el socio espera son referenciados por *myRole*. Cada *rol* provee un *portType* que representa el punto donde se puede acceder a un servicio ofrecido por un servicio Web.

Las *variables* definen el contexto del proceso de negocio y están constituidos elementos message descritos en la especificación WSDL. Estos mensajes contienen las operaciones de los *portTypes* involucrados en los roles de los *partnerLinks* relacionados entre el proceso y sus socios. Usualmente las variables se usan para enviar y recibir información con los socios. Aunque también se usan para mantener el estado del proceso el cual no se revela a los socios.

Las correlaciones se usan para identificar explícitamente una instancia de un proceso de negocio. Es un mecanismo a nivel de la aplicación que permite ligar los mensajes y conversaciones con las instancias de los procesos de negocio a los cuales han sido enviados.

Los *faultHandlers* se usan para cambiar el curso de una acción en caso de que haya ocurrido algún error durante la ejecución de un proceso. La forma en que se procede para capturar

```
<process name="...">
   <partnerLinks>
      <!– Lista de servicios con los
      cuales el workflow interactúa.
      Un conjunto de etiquetas partnerLink
      se incluye en esta sección
       -->
   </partnerLinks>

   <variables>
      <!– Lista de variables que mantienen
      el estado del proceso o de los
      mensajes que se envían a los socios.
      Un conjunto de etiquetas Variable
      se incluye en esta sección. -->
   </variables>

   <correlationSets>
      <!– Lista de conkuntos de correlación
      que sirven para definir un medio para
      identificar una conversación al nivel
      de la aplicación. Un conjunto de
      etiquetas correlationSet se incluye
      en esta sección.-->
   </correlationSets>

   <faulthandlers>
      <!– Lista de elementos para capturar
      los errores (catch o catchAll) -->
   </faulthandlers>

   <compensationHandler>
      <!– Un conjunto de actividades se
      incluye en esta etiqueta para
      establecer acciones de compensación.
      -->
   </compensationHandler>

   <eventHandlers>
      <!– Un conjunto de acciones activadas
      cuando un evento se recibe. -->
   </eventHandlers>
   <!– Actividades: Lógica de la ejecución
   de un Workflow (sequence, flow, receive,
   etc.).
   -->
</process>
```

Figura 2.5: Estructura general de un documento BPEL4WS.

los errores es parecida a la de Java. Un grupo de actividades *catch* son registradas y cada una de ellas captura un error específico y procede a ejecutar acciones posiblemente de compensación o de cambio en el flujo de la ejecución.

Los *compensationHandlers* se usan para establecer las actividades de compensación que deben ejecutarse. Como ejemplo podríamos suponer que un proceso requiere cancelar la compra de algunos boletos de avión. Entonces una orden de cancelación podría ponerse en un *compensationHandler* para regresar a un estado anterior al proceso.

En contraste, los *eventHandlers* se activan cuando un evento en particular ocurre. Las actividades que se ejecutan al interior de un *eventHandler* pueden ser de cualquier tipo excepto llamados a compensaciones. Los eventos se activan de dos formas: a la llegada de un mensaje o cuando un cierto tiempo ha ocurrido.

Además, BPEL4WS proporciona las actividades estructuradas que determinan la lógica en la ejecución del *workflow*. Estas actividades son:

- *sequence*: permite la ejecución de una o más actividades secuencialmente en orden de aparición.
- *switch*: permite el comportamiento condicional, es decir, divide el flujo de la ejecución de un proceso en ramas y toma alguna de ellas si se cumple una condición.
- *while*: permite la ejecución repetida de una actividad especificada hasta que una condición no sea válida.
- *pick*: espera la ocurrencia de uno de los eventos perteneciente a un conjunto de eventos y entonces realiza la actividad asociada con el evento.
- *flow*: ejecuta las actividades concurrentemente y sincronizadamente.

Finalmente, se cuenta con actividades básicas que permiten entre otras cosas el envío y la recepción de información entre el servicio Web orquestado y los servicios Web simples. Dentro de las actividades más importantes se encuentran:

- *receive*: este tipo de actividades especifica el *partnerLink* de donde se espera recibir la información, así como el *portType* y la operación que el socio debe ejecutar para comunicarse con el proceso. También especifica la variable usada para recibir la información proveniente del socio la cual es enviada por éste usando otro invoke o a través de un cliente SOAP. Esta actividad inicia la ejecución de un proceso.

- *reply*: en contraste a la anterior, esta actividad envía la respuesta a la petición recibida por receive. También especifica el *partnerLink*, *portType* y *operation* donde se debe ejecutar la respuesta. Mientras que la variable que se especifica dentro del *reply* contiene la información que se debe enviar como respuesta al solicitador del servicio.

- *invoke*: esta actividad permite la invocación síncrona (petición/respuesta) o asíncrona (invocación en una sola dirección) de una operación. En el primer caso hace una petición a un servicio Web u otro *workflow* y espera indefinidamente una respuesta. En el segundo caso no espera ninguna respuesta y continúa con su ejecución. También puede usarse como entrada de un *receive* y salida de un *reply*.

Para fijar ideas, tomemos un caso sencillo que ilustre el uso de algunas de las instruciones antes mencionadas. En este ejemplo se quiere enviar una cantidad n a un servicio Web orquestado. Este lo recibe y entonces envía el valor a un servicio Web simple que brinda el servicio de incrementar los valores que le lleguen. Una vez que el proveedor del servicio incrementa el valor, éste lo regresa al servicio Web orquestado quien a su vez se lo regresa al cliente. El código de dicho servicio Web orquestado sería como el que se muestra en la Figura 2.6

```
<process name="Invoke"
      targetNamespace="http://samples.otn.com"
      suppressJoinFailure="yes"
      xmlns:tns="http://samples.otn.com"
      xmlns:services="http://services.otn.com"
      xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
 <partnerLinks>
  <partnerLink name="client"
        partnerLinkType="tns:Invoke"
        myRole="InvokeProvider"
        partnerRole="InvokeRequester"/>
  <partnerLink name="IncrementService"
        partnerLinkType="services:IncrementService"
        partnerRole="IncrementServiceProvider"/>
 </partnerLinks>
 <variables>
  <variable name="input" messageType="tns:InvokeRequestMessage"/>
  <variable name="output" messageType="tns:InvokeResultMessage"/>
  <variable name="request"
         messageType="services:IncrementServiceRequestMessage"/>
  <variable name="response"
         messageType="services:IncrementServiceResponseMessage"/>
 </variables>
 <sequence>
  <!—Recibe la petición del cliente-->
  <receive name="receiveInput" partnerLink="client"
        portType="tns:Invoke"
        operation="initiate" variable="input"
        createInstance="yes"/>
  <!—Inicializa la entrada del IncrementService -->
  <assign>
   <copy>
    <from variable="input" part="payload" query="/value"/>
    <to variable="request" part="payload" query="/value"/>
   </copy>
  </assign>
  <invoke name="invoke" partnerLink="IncrementService"
       portType="services:IncrementService"
       operation="process" inputVariable="request"
       outputVariable="response"/>
  <assign>
   <copy>
    <from variable="response" part="payload" query="/result"/>
    <to variable="output" part="payload" query="/result"/>
   </copy>
  </assign>
  <!—Regresa el resultado al cliente -->
  <reply name="replyOutput" partnerLink="client"
       portType="tns:InvokeCallback"
       operation="onResult" Variable="output"/>
 </sequence>
</process>
```

Figura 2.6: Ejemplo simple de la forma en que se utiliza BPEL4WS.

El ejemplo de la Figura 2.6 funciona de la siguiente manera. Primeramente, se establecen los *PartnerLinks* que representan tanto al cliente que realiza la petición (*client*) como el servicio Web que provee el servicio de incremento de valores (*IncrementService*). Después se establecen las variables de entrada y salida del servicio Web orquestado (*input* y *output* respectivamente) y del servicio Web que ofrece el servicio de incrementar el valor (*request* y *response*). Posteriormente, se establece una secuencia de actividades a través de la sentencia *sequence*. La primera actividad es recibir del cliente el valor que desea incrementar. Esto se logra poniendo la actividad *receive* cuyo nombre es *receiveInput* dentro de la cual se establece el *partnerLink* que determina la entidad de dónde se recibe la información (en este caso es *client*). También se especifica la variable que guardará el valor n que en este caso es *input*. Después se copia el valor de entrada *input* a la variable de entrada *request* del servicio Web que provee el servicio de incremento (*IncrementService*) a través de la actividades *assign* y *copy*. Posteriormente, se invoca el servicio *IncrementService* pasándole como entrada el valor contenido en *request* que es el mismo que el valor n enviado por el cliente. El resultado de la invocación se almacena en la variable *response* que contiene el valor incrementado. Entonces, se copia este valor en la variable de salida *output* a través de un *assign* y un *copy*. Finalmente, se regresa una respuesta al cliente a través del *reply* con el valor incrementado que se encuentra contenido en la variable *output*.

En esta sección se ha descrito con mayor detalle la tecnología relacionada con los servicios Web. A continuación se expone un enfoque promovido para dotar de mayor "inteligencia" a los servicios Web.

2.2. Un enfoque semántico a los servicios Web

La mayoría de la información que actualmente se encuentra disponible en Internet es únicamente comprensible por seres humanos. Esto impide sustancialmente la creación de aplicaciones capaces de realizar tareas autónomamente. La Web semántica se puede ver como una solución a este problema. Esta permitirá la creación de un repositorio de información inteligible por las computadoras transformando a la Web en una base de conocimientos. Es necesario, por tanto, desarrollar lenguajes formales y mecanismos de inferencia para representar la información y razonar sobre ella.

El DARPA Agent Markup Language for Services (DAML-S) es el primer paso que se ha dado para definir un lenguaje de este tipo. En esta sección se describe DAML-S y la relación que tiene con otros trabajos relacionados con esta tesis.

2.2.1. Resource Description Framework

El Resource Description Framework (RDF) [3] es un lenguaje que puede usarse para anotar el contenido de una página Web. RDF es muy similar a una gráfica dirigida simple. Esto lo hace un lenguaje con una gran simplicidad arriba del cual cualquier otro tipo de método de modelación de información puede ponerse.

RDF provee una infraestructura que permite codificar, intercambiar y reutilizar información estructurada. RDF se basa en XML quien lo dota de la posibilidad de crear métodos sin ambigüedad. RDF permite además la publicación de vocabularios comprensibles para las personas y procesables por las máquinas. Estos vocabularios promueven la reutilización y la extensión de la semántica de los meta datos entre diferentes comunidades.

Sin embargo, debido a que RDF es limitado para expresar tipos de datos, enumeraciones, etc. se ha visto la necesidad de un lenguaje que permita describir clasificaciones y propiedades de los recursos de una manera más sofisticada que RDF. DAML-S surgió como un lenguaje más expresivo que extiende las capacidades de RDF y con más posibilidades de abarcar los alcances previstos por la Web semántica.

2.2.2. DAML-S

DAML-S [24] es un lenguaje ontológico basado en DAML+OIL orientada a los servicios Web. El objetivo principal de DAML-S es describir las propiedades y capacidades de los servicios Web de tal forma que sean interpretables por las computadoras. Lo anterior permite el descubrimiento de servicios Web específicos, su invocación, composición automática con otros servicios, verificación de las propiedades del servicio y el correspondiente monitoreo de las tareas ejecutadas por los servicios.

Básicamente, DAML-S provee a los servicios Web de una descripción semántica en la capa de la aplicación que describe lo que un servicio puede hacer y no sólo cómo lo puede hacer [24]. Esta capa se ubica encima de WSDL y complementa la estructura actual de los servicios Web (ver Figura 2.1). En la Figura 2.7 se muestra la estructura de los servicios Web usando DAML-S, en donde se aprecia que las tres capas inferiores de la estructura de los servicios Web quedan intactas. Sin embargo, la capa de orquestación y de consulta y descubrimiento de servicios Web se ha modificado. Un servicio Web en DAML-S se describe con:

- El perfil del servicio: Provee una vista de alto nivel de un servicio Web, es decir, es el análogo de UDDI. Sin embargo, a diferencia de UDDI, el perfil de servicio permite la representación de capacidades de un servicio Web. Otra diferencia notable con

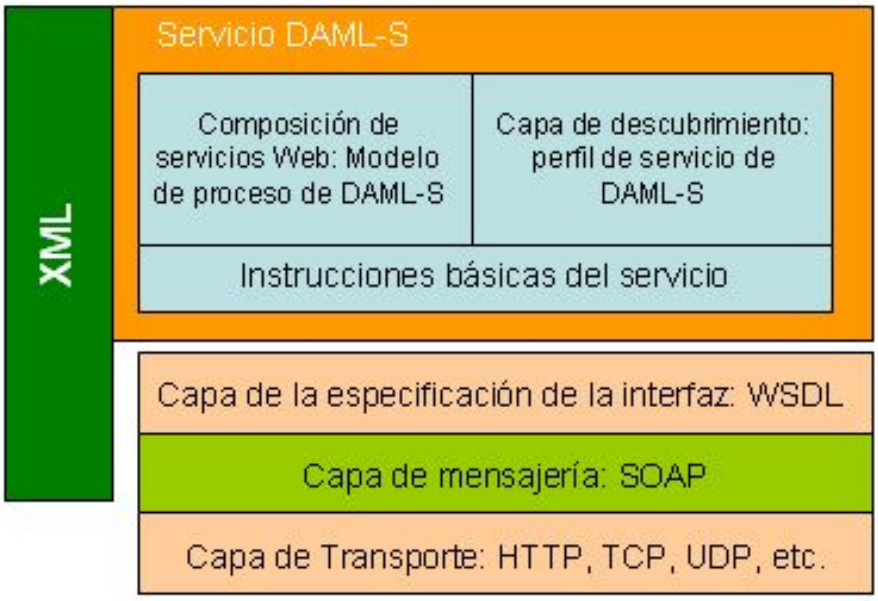

Figura 2.7: Arquitectura de los servicios Web usando DAML-S.

UDDI es que el perfil del servicio no describe los puertos implementados por un servicio Web, sino por la capa de las instrucciones básicas del servicio.

- El modelo del proceso: Muestra las tareas que un servicio Web realiza, el orden de su ejecución y las consecuencias de cada una de ellas. Es similar a BPEL4WS aunque pone especial énfasis en los efectos de la ejecución de un servicio. Es preciso mencionar que esta capa aún está en elaboración y hasta el momento no se dispone completamente de ella.

- Las instrucciones básicas del servicio: Enlaza la descripción abstracta de los envíos de información de un servicio Web con una operación WSDL.

DAML-S actúa como un medio para definir ontologías usando los constructores basados en DAML+OIL que definen el concepto de un servicio Web. También actúa como un lenguaje que describe servicios Web específicos que los usuarios y otros servicios pueden

descubrir e invocar usando SOAP y WSDL [25].

Entonces, los servicios Web que usan DAML-S toman a UDDI, WSDL y SOAP para descubrir otros servicios e interactuar con ellos, y éstos usan a DAML-S para integrar estas interacciones en el problema que desean resolver.

De esta manera DAML-S permite a los servicios Web contar con representaciones de contenido formales y con razonamiento sobre sus interacciones y capacidades. A continuación se muestran los trabajos relacionados con esta tesis y los enfoques que dan para resolver problemas expuestos en este trabajo. Algunos de ellos usando DAML-S como eje principal en su solución.

2.3. Trabajos relacionados

En la literatura se ha podido observar el gran interés y preocupación que se tiene por crear servicios composicionales capaces de interactuar dinámicamente con otros servicios Web. La importancia de este problema radica en la naturaleza inherentemente cambiante de los procesos internos y externos de una organización. Los participantes en un proceso pueden cambiar de un momento a otro en función de las condiciones del ambiente.

Para ejemplificar lo anterior, supongamos que una empresa requiere hacer un pedido de productos para equilibrar sus niveles de inventario. La empresa busca en la red a través de un agente, una tienda que pueda satisfacer esta petición. El agente encuentra un conjunto de tiendas que potencialmente satisfacen la petición (esto se conoce como *semantic matching*). El agente escoge la mejor opción de acuerdo a un criterio específico como el precio más bajo o el tiempo de entrega más corto y le realiza la petición. Sin embargo, por alguna razón, la tienda experimenta problemas técnicos que imposibilitan

a la tienda dar el servicio. El agente advierte el problema y procede a realizar la petición a alguna de las otras empresas. Finalmente, el agente obtiene el resultado de la petición y avisa al cliente que el pedido será cubierto por la tienda.

Se puede apreciar en el ejemplo anterior que en principio, el conjunto de tiendas que el agente busca es completamente desconocido. Una vez que encuentra alguno, lo puede integrar dinámicamente al proceso de negocio que lo requiere; en este caso es el nivel de inventario de una empresa. Muchos son los procesos de negocio que presentan situaciones similares en las que es necesario integrar dinámicamente los socios comerciales a dichos procesos.

Diversos han sido los enfoques que se han propuesto para resolver este problema. A continuación se presentan algunos de los más importantes.

2.3.1. Invocación dinámica usando DAML-S

En [26] se describe la necesidad de considerar restricciones de dominio y dependencias entre servicios para seleccionar servicios legales que se enlacen con un *workflow*. En este artículo se presenta un sistema que dinámicamente enlaza servicios Web con un proceso de negocio mediante un mecanismo de descubrimiento semántico basado en restricciones.

La necesidad de un sistema como éste radica en el hecho de que la actual especificación de BPEL4WS sólo permite agregar socios a un proceso de negocio en tiempo de diseño. Además de que WSDL carece de expresividad semántica para poder deducir las capacidades de un servicio Web. Por estas razones se hace uso de un lenguaje semántico más expresivo como DAML-S. Este lenguaje les ha permitido dotar a su sistema del descubrimiento automático de servicios.

El sistema consiste de un *proxy* genérico que se vincula con nodos específicos en un *workflow* descrito en BPEL4WS. Este *proxy* toma los requerimientos (de dominio o de dependencia de servicio) en forma de restricciones escritas en DAML-S provenientes del *workflow* y que representan a un *subworkflow*. Entonces, el *proxy* descubre los servicios necesarios que se encuentran en un repositorio semántico UDDI. A continuación genera un conjunto de servicios Web que satisfacen las restricciones de servicios mediante un módulo que hace *semantic matching*. Después pasa los servicios a un módulo llamado *Constraint Checker* que se encarga de crear un nuevo conjunto de servicios compatibles con las restricciones de dominio. Finalmente, este conjunto de servicios se envían a un módulo que los invoca y regresa el control al *workflow* que llamó al *proxy*.

2.3.2. Coreografía de procesos de negocio para la colaboración B2B

En [27] se muestra que debido a la creciente complejidad de los ambientes de negocio, resulta de gran importancia desarrollar mecanismos que permitan incorporar procesos de negocio existentes a la lógica del negocio.

En este artículo se propone una metodología que permita crear procesos de negocio colaboradores. Esta metodología ha sido diseñada con el fin de representar patrones de interoperabilidad entre dos procesos de negocio y para automatizar dichos patrones sistemáticamente. Además, la metodología propone establecer procesos de negocio contractuales y ejecutables que usen un protocolo interfaz que los vincule. Los procesos contractuales tienen la finalidad de describir cómo asociar los procesos internos de los socios al *workflow* existente. Los procesos internos de los socios se conocen como procesos ejecutables.

En este enfoque se busca reutilizar los procesos ejecutables e incorporarlos sistemática-

mente a otro proceso a partir de la definición de patrones claramente identificados en el Workflow Management Coalition (WfMC) [WfMC]. De hecho, en [27] se han hecho extensiones a dichos patrones identificando seis de ellos.

Este trabajo es similar al nuestro en el sentido de que también intenta atacar el problema de la reutilización de *workflow* a partir de patrones para incorporarlos a la lógica de negocio. Sin embargo, difiere en que está enfocado a incorporar *workflows* o parte de un *workflow* a otros *workflows* para que reutilicen una serie de pasos que se encuentran en un *workflow* que resultan ser útiles dentro de la lógica de otro negocio. Mientras que el nuestro tiene como finalidad agregar dinámicamente servicios Web a la lógica de un proceso de negocio.

2.3.3. Composición de *workflows* usando ontologías de servicios Web semánticos

En [28] se muestra cómo Internet está pasando de ser un simple repositorio de texto e imágenes a un proveedor de servicios. En este sentido, la Web será más dinámica y por tanto la necesidad de integrar los servicios Web dinámicamente a los *workflows* será mayor.

En este artículo se presenta un agente basado en DAML-S, capaz de construir *workflows* a partir de ontologías de servicios Web. Este agente usa descripciones semánticas de los servicios Web para encontrar algunos que coincidan con los requerimientos de un *workflow*. Se hace especial énfasis en el problema de que un servicio Web no se encuentre disponible al momento de la ejecución del *workflow*. En este caso, es necesario substituirlo por otro cuya funcionalidad sea altamente similar al que se buscaba originalmente.

El modelo que se utiliza en este trabajo consiste en un agente que recupera de un repo-

sitorio de *workflows*, el *workflow* que cumple con sus requerimientos; y de un repositorio semántico UDDI los servicios Web que cumplen con ciertos criterios. Entonces el *workflow* se ejecuta y se esperan los resultados.

El trabajo resuelve principalmente dos casos en los que es necesario crear o actualizar dinámicamente un *workflow* de servicios Web: reemplazar un servicio Web en un *workflow* existente por otro que tenga una funcionalidad similar; y definir un nuevo *workflow* usando los servicios Web que se encuentran disponibles en la Web. En el primer caso se presenta como solución modificar únicamente el WSDL donde se hace referencia a los anteriores servicios Web por los nuevos. El segundo caso no es aún resuelto.

Aunque en este artículo no se muestra claramente la forma en que se recuperan los *workflows* o incluso la estructura del repositorio de *workflows*, se puede apreciar que el trabajo es muy similar al presentado en esta tesis. Sin embargo difiere del nuestro en que nuestra propuesta no utiliza la Web semántica para dar un solución al problema.

2.4. Conclusiones

En conclusión, se puede apreciar que los diferentes trabajos relacionados son similares al nuestro en el sentido de que buscan resolver el problema de agregar dinámicamente socios a un proceso de negocio. Sin embargo, la forma en que abordamos el problema difiere sustancialmente.

En el trabajo presentado en la sección 2.3.1, se plantea un *proxy* que se encarga de seleccionar los servicios Web adecuados, invocarlos y obtener los resultados. A pesar de que este enfoque resuelve parcialmente el problema de una forma elegante, resulta ser un tanto ineficiente para problemas reales ya que el *proxy* se encarga de coordinar todo tipo

de interacciones que ocurren en el proceso de negocio. Imaginemos el caso en el que una gran cantidad de usuarios ejecutan el mismo *workflow* que llama repetidamente al *proxy*. Entonces una gran cantidad de llamados al *proxy* serían enviados, lo que provocaría un *cuello de botella*. Esto limitaría significativamente su rendimiento ya que el *proxy* tendría que interactuar con cada instancia del *workflow*. En lugar de dejarle la responsabilidad a un *proxy* de interactuar con el *workflow* y con los servicios Web, hemos desarrollado un sistema (Administrador de Procesos de Negocio) que, bajo una petición de un cliente, construye un *workflow* quien se encarga de realizar el proceso de selección de los servicios Web adecuados, invocarlos y obtener los resultados. Esto evita una carga de trabajo excesiva a una sola entidad ya que en nuestra propuesta se producen *workflows* por separado que se ejecutan independientemente y más eficientemente debido a que no se recurre a un invocador central.

Por otro lado, en el artículo de la sección 2.3.2 se hace especial énfasis en la integración parcial o total de *workflows* a la lógica de un negocio. De hecho, se propone una arquitectura y metodología completa para conseguir este objetivo. Sin embargo, la principal aplicación de dicho trabajo es el comercio *Business to Business* ya que, como se explicó, los autores proponen la identificación de patrones de *workflows* que son recurretes al interior de las organizaciones y plantean formas de integrarlos parcial o totalmente en tiempo de ejecución a la lógica de un negocio. En contraste, nuestra propuesta tiene relación hasta el momento sobre todo con el comercio *Business to Customer*. En ella se plantea la integración dinámica exclusivamente de servicios Web a procesos de negocio.

Por último, en el artículo de la sección 2.3.3 se muestra un agente basado en herramientas semánticas que realiza un proceso similar al nuestro para agregar dinámicamente servicios Web a un proceso de negocio. Sin embargo, no han resuelto el problema de generar dinámicamente un nuevo *workflow* según las necesidades que se tengan. En nuestra propuesta sí abordamos el problema de generar un nuevo *workflow* y proponemos una solución a

través de una metodología que se explica en los siguientes capítulos. En esencia, nuestro enfoque es más simple aunque bastante útil debido a que resuelve el problema de añadir dinámicamente los servicios Web a un proceso de negocio modificando el *workflow* según sean las necesidades del cliente, sin necesidad de aplicar los lenguajes de marcado semántico. De hecho, la decisión de seleccionar el servicio Web más apropiado se realiza en parte dentro del *workflow* mediante la aplicación de algoritmos y en parte a la libre elección del cliente. Además, en nuestro trabajo no buscamos reemplazar las funciones de BPEL4WS por un lenguaje semántico como DAML-S, sino que extendemos las funciones de BPEL4WS al escribir los documentos BPEL4WS en formas de plantillas, concretizándolas a través de un sistema encargado de realizar esta tarea. Esto resulta mejor ya que las implementaciones de DAML-S aún no están terminadas y no han sido ampliamente probadas lo que implica riesgos al momento de ponerlas en funcionamiento en situaciones reales. En contraste, BPEL4WS ha sido ampliamente probado y goza de una importante aceptación y soporte de grandes empresas desarrolladoras de *software* como IBM o Microsoft por mencionar algunas.

Capítulo 3

El Administrador de Procesos de Negocio

Este capítulo tiene como objetivo presentar el tipo de problemas que el Administrador de Procesos de Negocio (APN) resuelve. En la primera parte se muestra el entorno del sistema. En la segunda parte se plantea un caso de estudio en el que se trata el problema de agregar dinámicamente un conjunto de socios comerciales conocido únicamente en tiempo de ejecución a un proceso de negocio. Posteriormente, se expone la solución propuesta al problema. Esta solución consiste en una metodología que propone una serie de pasos que resuelven parcialmente el problema. Por último, se muestran las interfaces de usuario desarrolladas que permiten a un cliente interactuar con el *workflow*. Estas interfaces están basadas en ontologías que facilitan la integración de socios comerciales a la cadena de suministro.

3.1. El contexto del APN

El APN y su entorno se ilustran en la Figura 3.1. Este es un modelo arquitectónico que muestra la interacción del APN con otras entidades. Se puede ver a partir de la figura que el APN interactúa directamente con un portal y la máquina BPEL4WS. El portal sirve esencialmente de enlace entre el cliente y el APN, esto es, una petición del cliente se hace a través del portal quien se encarga de comunicársela al APN.

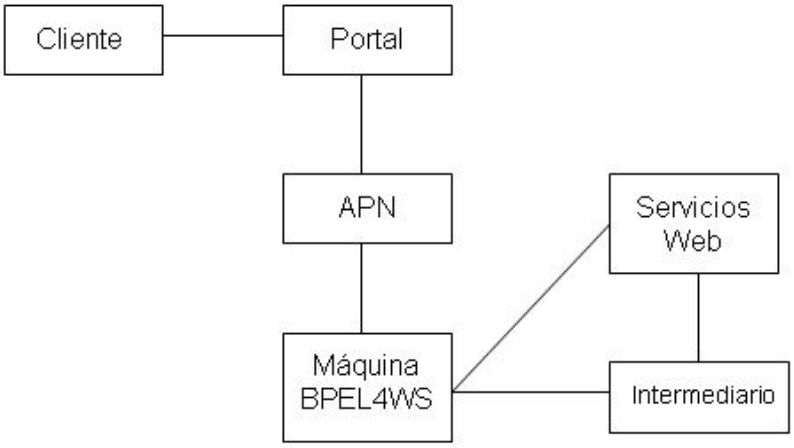

Figura 3.1: Modelo de contexto del APN

Por otro lado, el APN se conecta con la máquina BPEL4WS quien se encarga de proveer el servicio de ejecución de *workflows* que el APN le proporciona. A su vez, la máquina BPEL4WS se comunica con un Intermediario quien le provee información sobre los servicios Web expuestos por diversas tiendas. Eventualmente, la máquina BPEL4WS se comunica directamente con dichos servicios Web quienes ofrecen sus servicios de consulta y compra de productos por Internet.

3.2. Caso de estudio

La forma en que comúnmente se procede para orquestar una serie de servicios Web es, primeramente, determinar en tiempo de diseño un *workflow* en el que se establecen todas las relaciones que se tienen con otros servicios Web y posteriormente ejecutarlo. En este esquema los *workflows* se encuentran completamente determinados toda vez que se conocen de antemano los socios comerciales con los que se va a trabajar. Sin embargo, este esquema puede resultar un tanto rígido para cierto tipo de aplicaciones en las que el conjunto de socios con los que se va a tener contacto no pueden ser conocidos antes de la ejecución del *workflow*.

Supongamos el siguiente escenario:

- Una persona dedicada a la venta de aparatos electrónicos requiere satisfacer un pedido de n computadoras.
- Para conseguir las n computadoras a un precio razonable, esta persona recurre a nuestro portal en el que se tiene un conjunto de ontologías dentro de las cuales selecciona alguna (UN-SPSC),
- Selecciona entonces un servicio (precio más bajo) y
- Selecciona un producto (Computadora) y su cantidad.

La información anterior se envía a un sistema de intermediación [29] que cuenta con un conjunto de servicios Web ($get\_PriceandDeliveryTime$, $get\_ProviderQuantity$ y $get\_ProviderURLBuy$) que se encuentran en un nodo UDDI privado y que pueden ofrecer el servicio requerido para el producto seleccionado. En la Figura 3.2 se muestra el proceso de compra de una computadora y la interacción existente entre el *workflow* y los servicios

Web ofrecidos por el Intermediario. El sistema de Intermediación ofrece entonces al cliente un conjunto S de servicios Web que potencialmente pueden cubrir las necesidades del cliente. Esto lo hace a través del servicio Web *get_PriceandDeliveryTime* que brinda información acerca del precio y tiempo de entrega de un producto a partir de su código.

Del conjunto S se obtiene entonces un subconjunto S' que representa a los servicios con el precio más bajo, es decir, que cumplen con el criterio de búsqueda. Al interior del *workflow* se aplica un algoritmo de optimización que determina el conjunto S'.

Posteriormente, cada uno de los elementos del conjunto S' se pasa a los servicios Web *get_ProviderQuantity* y *get_ProviderURLBuy* quienes se encargan de dar información acerca de la cantidad de productos disponibles y el lugar donde pueden comprarse respectivamente. Esta información es de gran utilidad porque permite determinar si la cantidad de productos satisface la necesidad del cliente.

El cliente escoge entonces las tiendas en las cuales desea hacer su compra del conjunto S' y la cantidad de productos que desea comprar en cada una de ellas y forma un subconjunto S" del conjunto S'. Envía entonces una petición de compra a cada una de las tiendas pertenecientes al conjunto S".

Del ejemplo anterior se puede ver claramente que el conjunto S" no se puede determinar de antemano y entonces el actual esquema de construcción de *workflows* usando BPEL4WS no es suficiente para resolver problemas de este tipo. El enfoque que se ofrece en la presente tesis consiste en construir en tiempo de ejecución el documento BPEL4WS donde se incluyan dinámicamente los socios comerciales de un proceso de negocio.

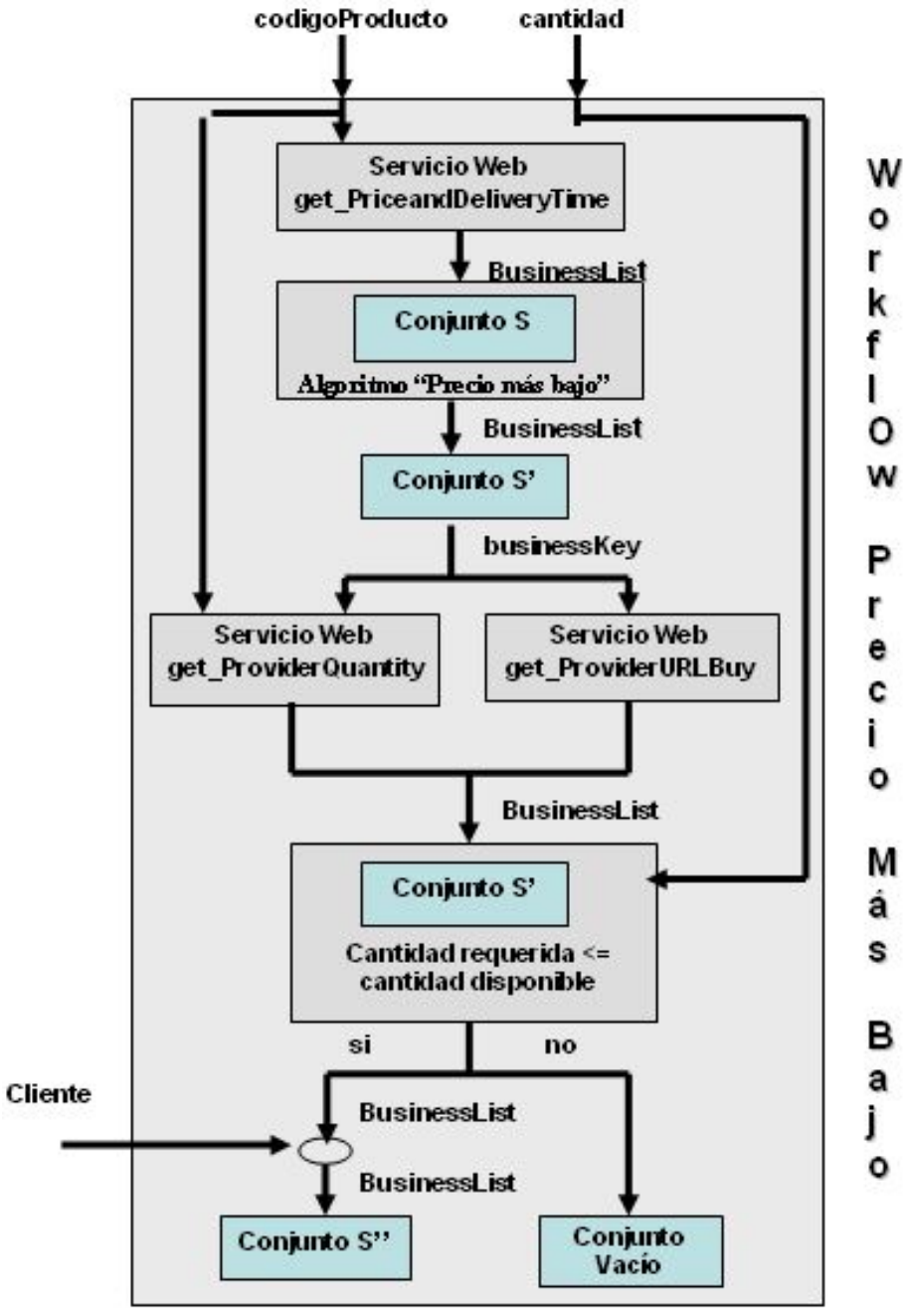

Figura 3.2: Estructura general de un *workflow* ejecutado para recuperar el precio más bajo de una computadora.

3.3. Funcionamiento general

En esta sección se presenta la metodología seguida para resolver el problema planteado en la sección anterior.

3.3.1. Descripción de procesos genéricos

Un proceso genérico de negocio es una abstracción de un proceso de negocio que comúnmente se lleva a cabo al interior o exterior de una organización. En esta abstracción se pretende diferenciar las partes de un proceso de negocio que no cambian de las que sí lo hacen. La forma en que se procede para crear las plantillas o procesos de negocio genéricos es, primeramente, construir un *workflow* funcional en BPEL4WS. Posteriormente, se analiza cuáles son las partes del *workflow* que pueden cambiar en tiempo de ejecución (como los socios comerciales) y cuáles son inherentes al proceso. Las partes que cambian se dejan incompletas obteniendo así una plantilla con base en las partes del *workflow* que no cambian. Estas partes serán completadas en tiempo de ejecución cuando el cliente decida lo socios comerciales con los que desea interactuar.

En el presente trabajo se han identificado claramente las partes de un proceso escrito en BPEL4WS que resultan similares cuando se intenta resolver un problema de la misma clase. Uno de estos problemas es la compra de un producto en una o más tiendas virtuales. Como se ha dicho, BPEL4WS no permite agregar dinámicamente socios a un proceso comercial. Por lo que si se desea comprar un producto en un conjunto variable de proveedores, resultaría ineficiente tener que hacer un documento BPEL4WS para cada uno de los proveedores. La idea radica en hacer una plantilla o proceso genérico que sea completada dinámicamente con un conjunto de tiendas.

Asimismo, basándonos en la suposición de que la información disponible de las tiendas ha seguido estándares como RosettaNet [30], se puede suponer que el algoritmo de compra de un producto es el mismo cada vez que se desea comprar un artículo del mismo tipo. En este sentido, una empresa interesada en implementar un software capaz de hacer compras en Internet, puede utilizar el algoritmo usado en este trabajo y adecuarlo a sus necesidades. Esto ayudaría enormemente en el desarrollo de software.

3.3.2. Recuperación dinámica de *workflow*s genéricos escritos en BPEL4WS

La solución propuesta consiste en primeramente construir un repositorio de plantillas BPEL4WS que describan de manera genérica las actividades comerciales que se pueden llevar a cabo dentro del portal. El diseño del repositorio se muestra detalladamente en el siguiente capítulo. Basta decir en esta sección que el APN recupera del repositorio la información necesaria para ubicar y obtener las plantillas BPEL4WS y WSDL.

Retomando el ejemplo de la compra de la computadora en alguna de las tiendas, el cliente puede estar interesado en comprar la computadora en la tienda que ofrezca el precio más bajo, o el tiempo de entrega más corto. De hecho, otra contribución de este trabajo es el uso de algoritmos de optimización en la composición dinámica de *workflows* organizacionales. Si al cliente le interesa comprar una cierta cantidad de computadoras en la tienda o tiendas que ofrezcan el precio más bajo, entonces el APN recupera de una base de datos la ubicación de la plantilla BPEL4WS que aplica este criterio de compra. Dicha ubicación se determina concatenando la información de la ontología de servicios con el nombre del proceso seguida de la extensión. La ruta de la plantilla quedaría determinada por:

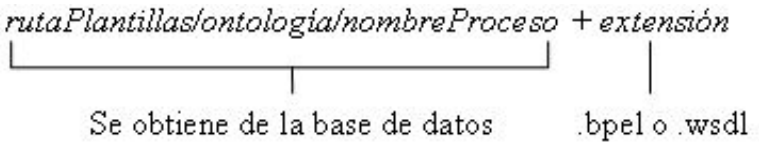

Figura 3.3: Ruta de la plantilla del proceso de negocio

Una vez que sabe dónde se encuentra la plantilla, el APN procede a recuperarla junto con la plantilla WSDL y las plantillas de configuración necesarias para la máquina virtual que interpreta los documentos BPEL4WS. Los documentos WSDL y BPEL4WS se obtienen usando las APIs de JAXP las cuales se encuentran en el Java Web Service Developer Pack [31]. A continuación se presenta el mecanismo de concretización de las plantillas.

3.3.3. Concretización de los *workflows*

En la sección anterior se explicó la forma en que el APN obtiene las plantillas que se usan para encontrar a los proveedores de servicios que ofrecen el producto requerido por el cliente. También se describió la forma en que se efectúa una consulta a la base de datos que contiene los documentos WSDL ofrecidos por el intermediario para recuperar los servicios Web apropiados para obtener el precio, tiempo de entrega, cantidad y punto de acceso. Estos servicios son $get\_PriceandDeliveryTime$, $get\_ProviderQuantity$ y $get\_ProviderURLBuy$ respectivamente y se basan en las ontologías de UN-SPSC y RosettaNet. Ahora se describe la forma en que son completadas dinámicamente las plantillas BPEL4WS y generadas las *Envolturas* (Wrappers) de los participantes en el *workflow*.

La implementación usada en este trabajo de la máquina de ejecución de BPEL4WS es el BPEL4WS Server de Oracle. Para poder agregar participantes en un proceso de negocio se requiere crear un documento XML llamado Wrapper por cada participante y cuyo nombre

de archivo es el nombre del servicio Web concatenado con la palabra Wrapper. Este documento permite establecer un enlace entre el proceso de negocio y sus participantes usando un *PartnerLinkType* para cada uno. Un ejemplo de este documento es la que se muestra en la Figura 3.4.

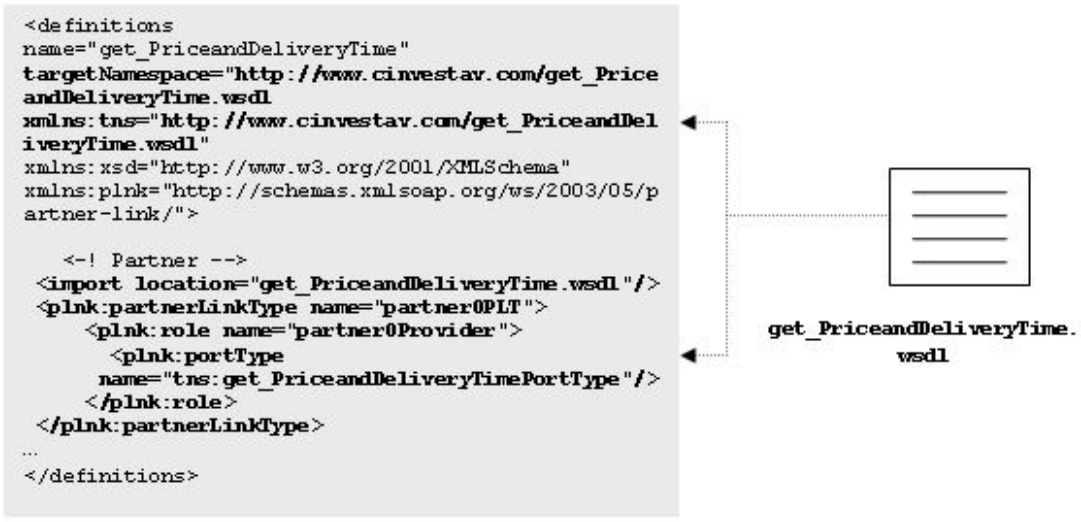

Figura 3.4: Wrapper correspondiente al servicio Web *get_PriceandDeliveryTime*.

Los documentos WSDL relacionados con un proceso de negocio son analizados por el APN y la información más importante se recupera y usa para generar el *Wrapper*. El espacio de nombres del servicio socio se incluye como atributo de la etiqueta *definitions*. En este caso se agrega el *targetNamespace* y el *xmlns:tns* con el espacio de nombres del servicio Web *get_PriceandDeliveryTime*. A continuación se agrega una sentencia *import* que contiene la localización del documento WSDL del participante. Finalmente, se agrega un *partnerLinkType* al elemento *definitions* junto con el correspondiente *role* y *portType*. Este último corresponde al ofrecido por el socio en su WSDL. El nombre del *portType* se divide en dos partes: el espacio de nombres y el nombre del *portType* del servicio remoto. Los *Wrappers* de los otros participantes (*get_ProviderQuantity* y *get_ProviderURLBuy*) en el proceso de negocio se construyen de la misma manera.

Por otro lado, el llenado del documento BPEL4WS ejecutable consiste en incluir ó modificar algunos de sus elementos. La Figura 3.5 muestra una plantilla BPEL4WS llenada. Evidentemente, en esta figura no se puede mostrar el código completo de la plantilla concretizada debido a que es bastante largo y resultaría confuso. Sin embargo, el código presentado es suficientemente ilustrativo para los fines de esta sección.

Como se muestra en la Figura 3.5, los espacios de nombre de los servicios Web, los *partnerLinks*, las variables de entrada y salida y las invocaciones a los servicios externos han sido agregados a la plantilla BPEL4WS. Nótese que un espacio de nombres y un *partnerLink* deben ser añadidos para cada servicio Web junto con sus variables de entrada y salida.

También se puede apreciar que a las variables se les asigna la información que el socio expone en su WSDL. De ahí se obtiene el tipo del mensaje con el que se va a intercambiar información (entrada y salida).

Por otro parte, los *partnerLinks* se construyen con la información que proviene del WSDL construido en la sección anterior. En particular, el nombre del *partnerLinkType* y sus correspondiente *partnerRole* deben coincidir con el que se agrega en el *partnerLink* del BPEL4WS. El nombre del *portType* asociado con el *partnerLinkType* también se obtiene del WSDL.

La descripción del proceso de negocio escrito en BPEL4WS se ejecuta en la máquina de *workflows* que se encarga de orquestar las invocaciones a los servicios Web, como se describe más detalladamente en la siguiente sección.

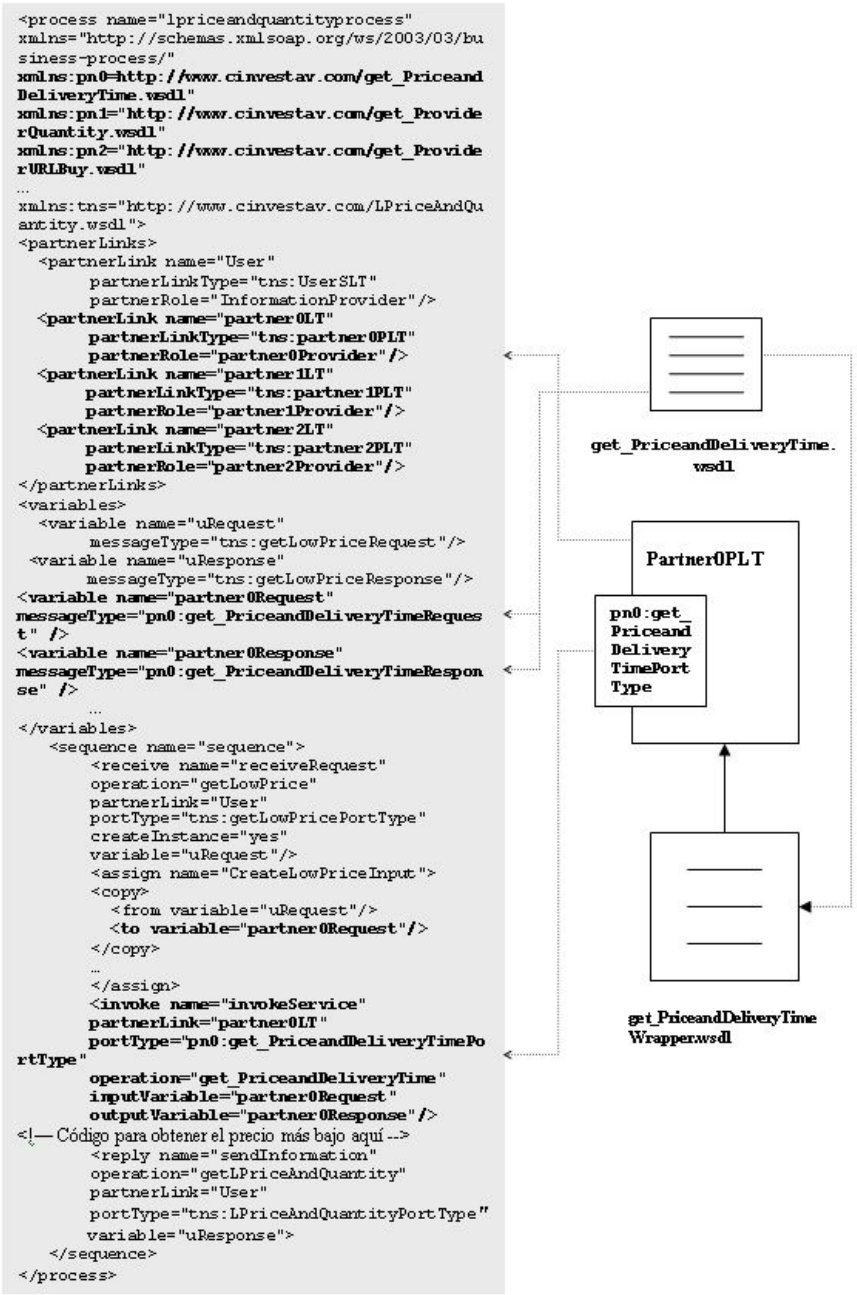

Figura 3.5: Plantilla BPEL4WS concretizada. Las instrucciones en negritas son añadidas o modificadas dinámicamente.

3.3.4. Ejecución del *workflow*

Una vez que el *workflow* se concretiza completamente, se procede a su ejecución. Las plantillas concretizadas se almacenan en una máquina BPEL4WS para su ejecución. En particular, para la máquina BPEL4WS de Oracle se usa el siguiente comando que ejecuta el *wokflow*.

$$obant - f\ /ruta/ServiceN/build.xml$$

donde N representa el número del identificador del usuario que corresponde al número de sesión que automáticamente se le asigna cuando se establece la comunicación con el servidor de aplicaciones Tomcat.

Por otro lado, para comunicarse con el *workflow*, el APN construye mensajes SOAP conteniendo la información proporcionada por el cliente. Siguiendo con nuestro ejemplo, el cliente envía al *workflow* el código de la computadora, la ontología y la cantidad en un mensaje SOAP. La ejecución del *worflow* es la que se muestra en las Figuras 3.6, 3.7, 3.8 y 3.9, si se desea concretizar la plantilla correspondiente al *Precio más bajo y cantidad mínima de productos*.

Los pasos que se muestran en las Figuras en forma de números encerrados en círculos se explican a continuación.

1. En la actividad **1** (*receive*), el cliente envía un mensaje SOAP al *workflow* donde especifica la información que solicita. El *workflow* inicia su ejecución en el momento en que el mensaje proveniente del cliente es capturado por la actividad *receive*. En este caso, el cliente inicia la ejecución del *workflow* enviando el código, la ontología y la cantidad del producto. Las flechas de entrada y salida que se muestran en la

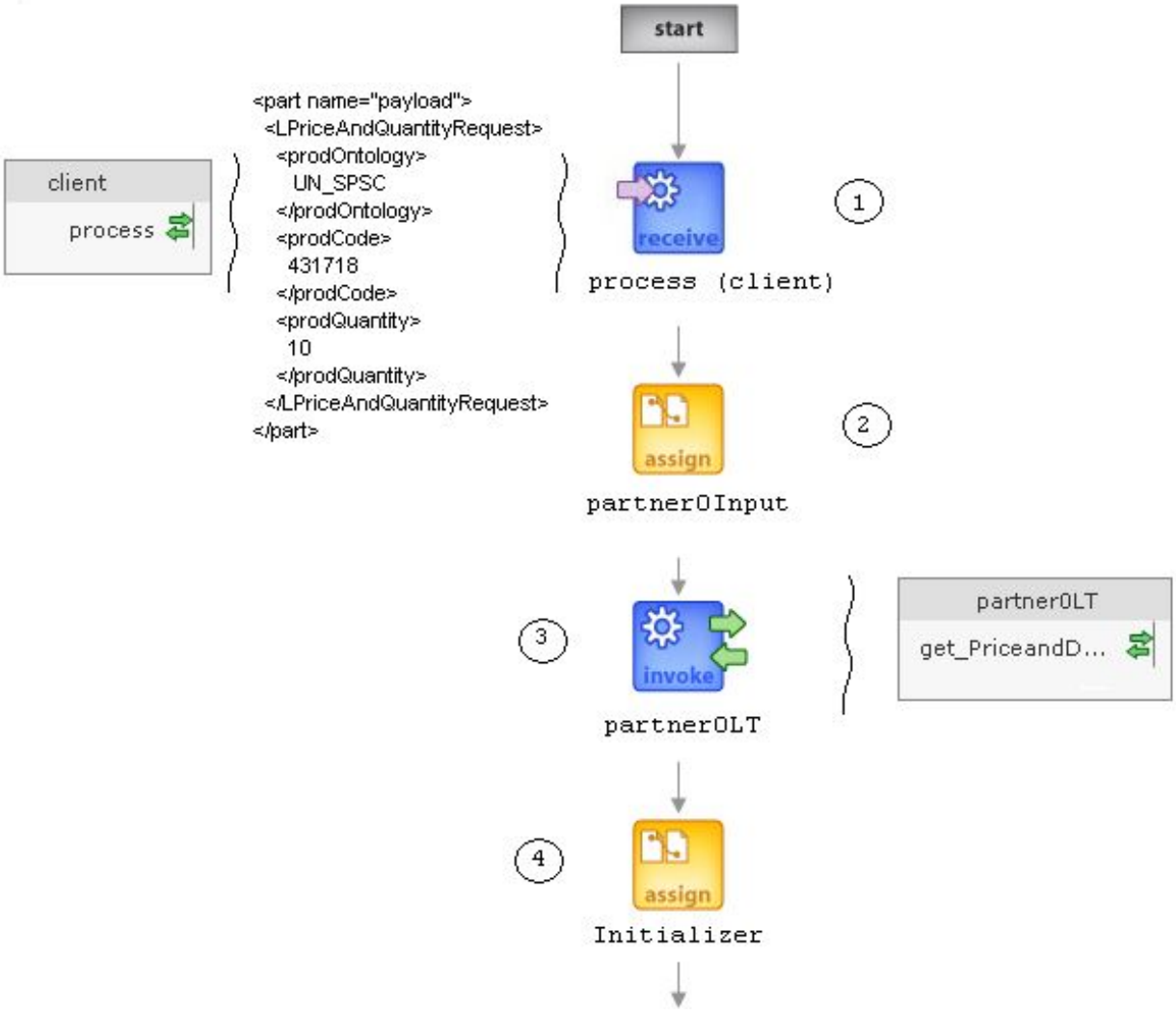

Figura 3.6: *workflow* del precio más bajo y cantidad mínima de productos Parte 1.

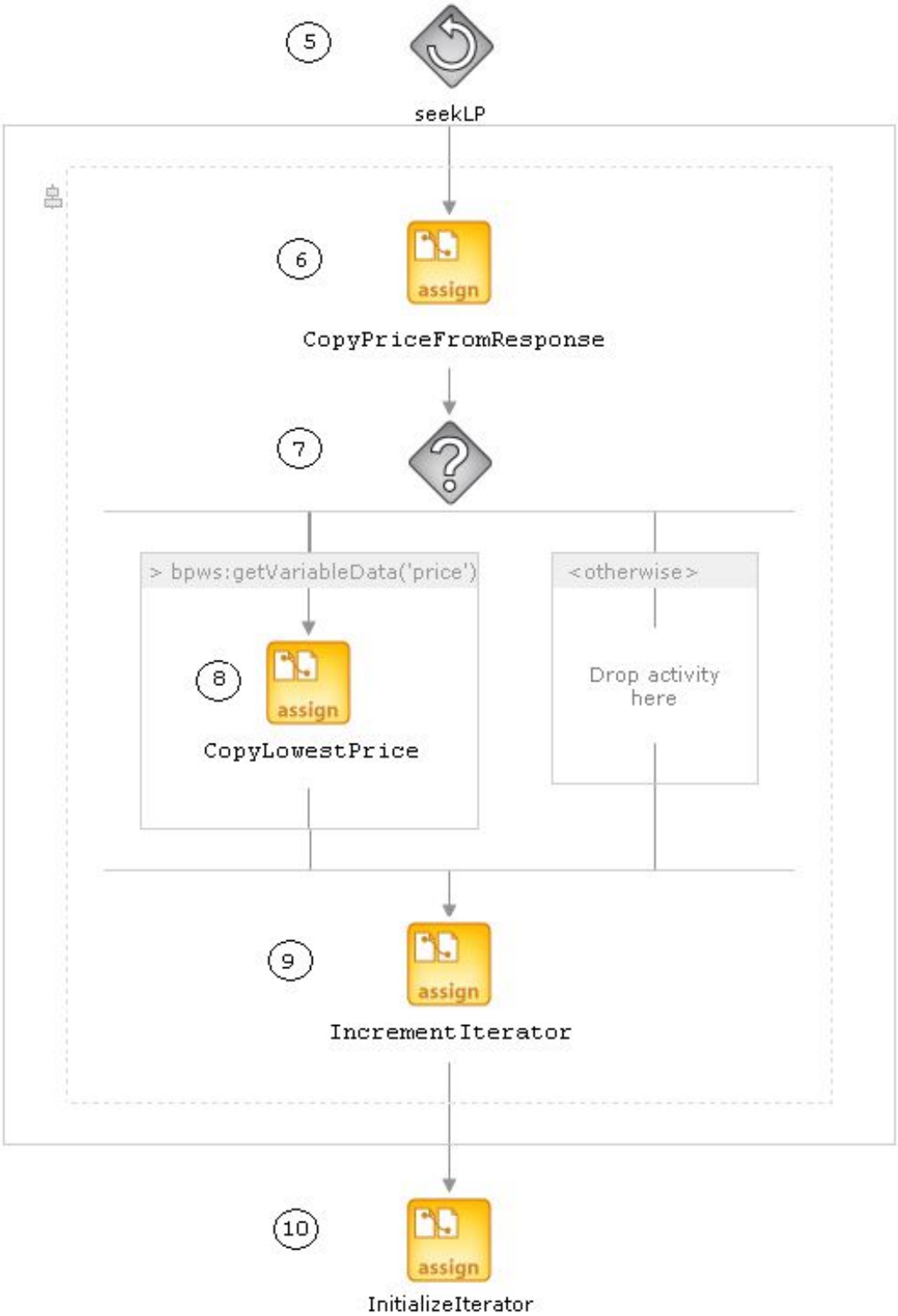

Figura 3.7: *workflow* del precio más bajo y cantidad mínima de productos Parte 2.

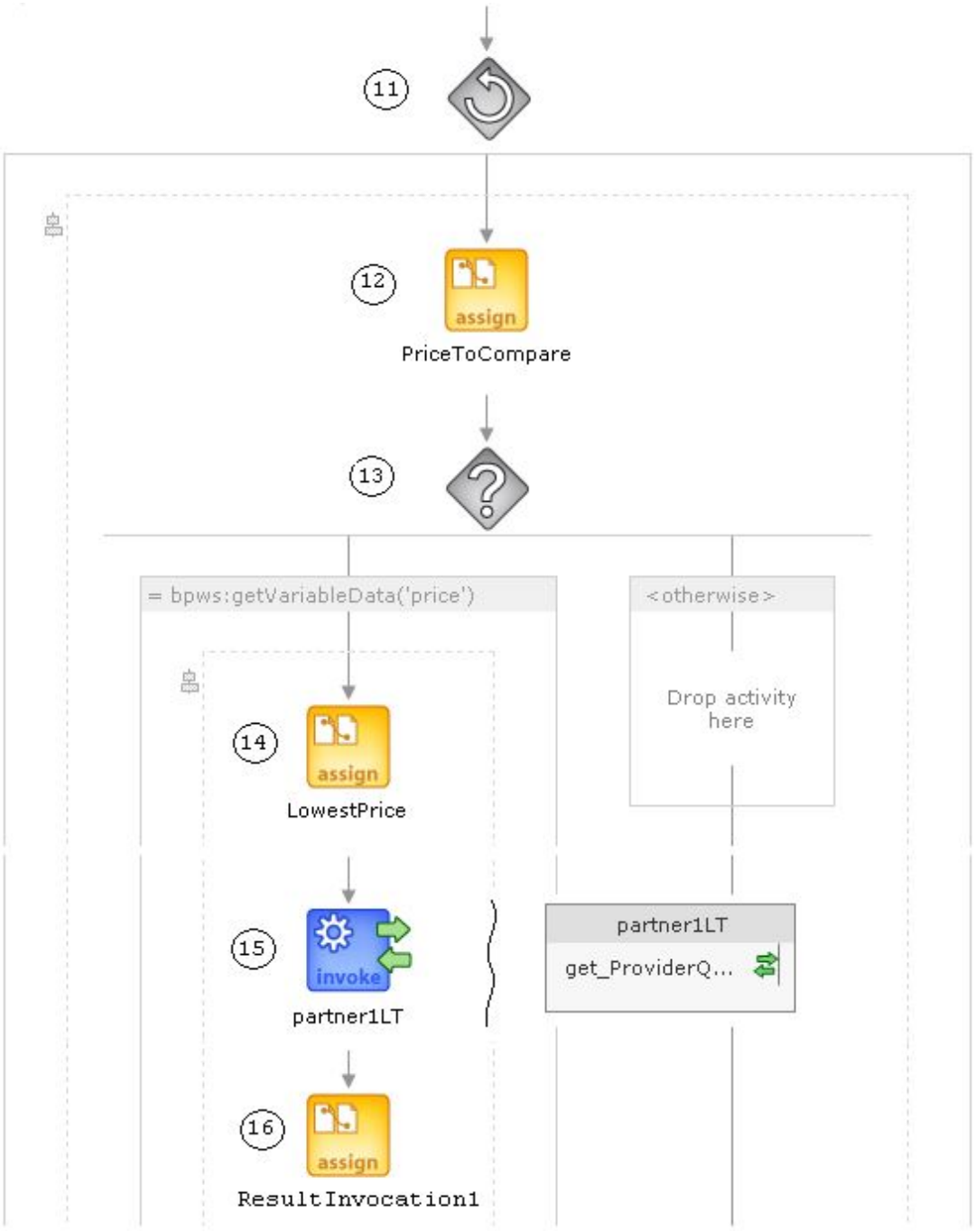

Figura 3.8: *workflow* del precio más bajo y cantidad mínima de productos Parte 3.

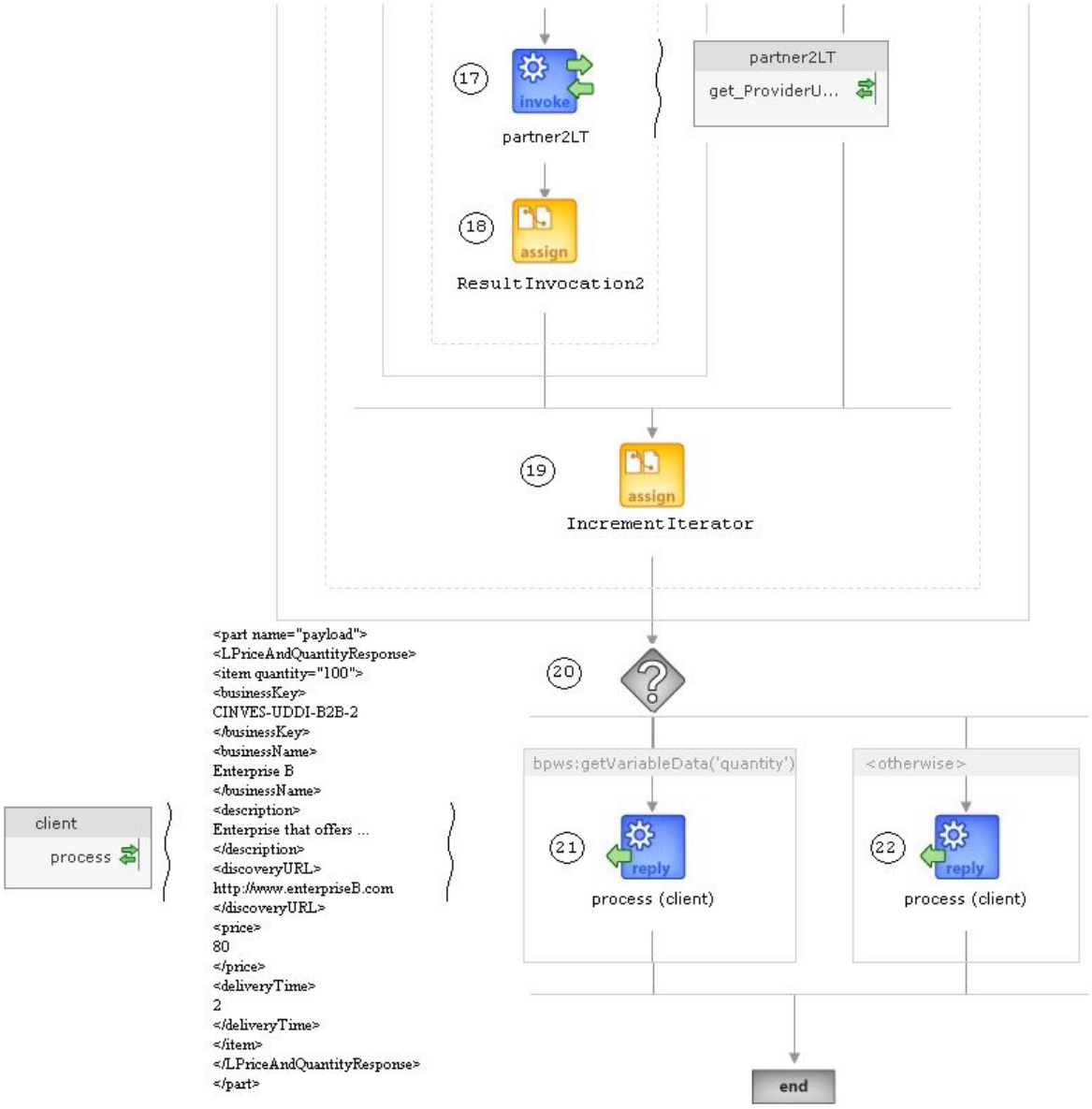

Figura 3.9: *workflow* del precio más bajo y cantidad mínima de productos Parte 4.

caja del cliente indican que la petición es síncrona. Así pues, el flujo desde donde se inició la petición queda bloqueado hasta que reciba una respuesta.

2. La actividad **2** (*assign*) copia parte de la información recibida, en las variables que sirven de entrada para invocar el primer proveedor de servicios ($get\_Priceand-DeliveryTime$). Este servicio requiere únicamente como entrada la ontología y código del producto.

3. La actividad **3** (*invoke*) invoca el servicio $get\_PriceandDeliveryTime$ de forma síncrona con la información recabada de la actividad **2**. Este servicio obtiene un conjunto de tiendas que ofrecen el producto junto con su precio y tiempo de entrega.

4. La actividad **4** inicializa variables locales como el iterador, contador, etc. que tienen el propósito de limitar el número de iteraciones del ciclo *while* de la actividad **5**. También copia el precio del producto que ofrece la primera tienda y lo propone como el precio más bajo asignándolo a la variable *lowestPrice*.

5. La actividad **5** (*while*) hace un número limitado de iteraciones n; donde n denota el número de tiendas del punto anterior. Su objetivo es determinar el precio más bajo dentro de las tiendas para el producto solicitado.

6. La actividad **6** copia el precio de la siguiente tienda en una variable local.

7. La actividad **7** (*switch*) compara el precio de la tienda con el precio más bajo que hasta el momento se tiene. Si es menor pasa a la actividad **8**, de lo contrario no realiza nada.

8. La actividad **8** asigna el precio del producto ofrecido por la i-ésima tienda en la variable *lowestPrice* y lo toma como el precio más bajo.

9. La actividad **9** incrementa el contador.

10. La actividad **10** inicializa nuevamente el contador para entrar al ciclo **11**.

11. La actividad **11** itera tantas veces como tiendas existan (obtenidas de la actividad **3**). Su objetivo es el de obtener las tiendas que ofrezcan el precio más bajo y pasarlas a los servicios *get_ProviderQuantity* y *get_ProviderURLBuy* para obtener su cantidad y punto de acceso respectivamente.

12. La actividad **12** obtiene el precio de la i-ésima tienda.

13. La actividad **13** compara el i-esimo precio con el más bajo encontrado en el ciclo *while* anterior. Si son iguales pasa a la actividad **14**, sino, no realiza nada.

14. La actividad **14** inicializa la variable de entrada del servicio *get_ProviderQuantity* con el *businessKey* de la tienda y el código y ontología del producto.

15. La actividad **15** invoca de forma síncrona el servicio *get_ProviderQuantity* pasándole en la variable de entrada del servicio Web el *businessKey* de la tienda y el código y ontología del producto. En la varible de respuesta de esta misma actividad se almacena información relacionada con el producto donde se encuentra su cantidad.

16. La actividad **16** asigna los resultados de la actividad anterior a la variable de salida de la actividad *reply*.

17. La actividad **17** invoca de forma síncrona el servicio *get_ProviderURLBuy* pasándole como entrada el *businessKey* de la tienda. El servicio regresa el punto de acceso de la tienda.

18. La actividad **18** asigna la respuesta de la actividad anterior a una variable que contiene la información que se va a regresar al cliente.

19. La actividad **19** incrementa el contador en 1.

20. La actividad **20** verifica si la cantidad de productos de la diferentes tiendas es cuando menos la solicitada por el cliente. Si es así, se procede a la actividad **21**, sino se pasa a la actividad **22**.

21. La actividad **21** envía en respuesta al cliente, mediante la actividad *reply*, la lista de tiendas que tienen el producto al menor precio y cuando menos la cantidad solicitada.

22. Opcionalmente, la actividad **22** envía al cliente una lista vacía que se interpreta como *Las tiendas no cumpen con los requisitos del cliente.*

Es preciso señalar que en la especificación 1.1 de BPEL4WS, no existen ciclos *for* ni instrucciones *if*, por lo que se han tenido que simular con las actividades *while* y *switch* respectivamente. Dicho lo anterior, se puede constatar que el orden de complejidad del algoritmo correspondiente al *workflow* es $O(n)$, donde n representa al número de tiendas que ofrecen el producto. También se puede ver que no existe forma de que el *workflow* entre en un *deadlock* ya que en caso de que no haya ninguna tienda con el producto, no se ejecuta ninguno de los ciclos *while* y si existe un número de tiendas, entonces los ciclos *while* realizan un número limitado de iteraciones correspondientes al número de tiendas que ofrecen el producto.

Una vez que el *workflow* se ejecuta exitosamente, envía de regreso al cliente la lista S'' de proveedores que satisfacen las condiciones impuestas por éste. Entonces, el *workflow* es desinstalado de la máquina BPEL4WS.

Después de que el cliente selecciona a los proveedores, una plantilla BPEL4WS para solicitar la orden de compra se recupera esta vez del repositorio, se concretiza y se ejecuta de la misma forma que se describió anteriormente por el APN. Las órdenes de compra son entonces enviadas en paralelo a los proveedores y las correspondientes respuestas de cada uno de los proveedores se reciben. El envío de solicitudes a las diferentes empresas se puede ver en la Figura 3.10.

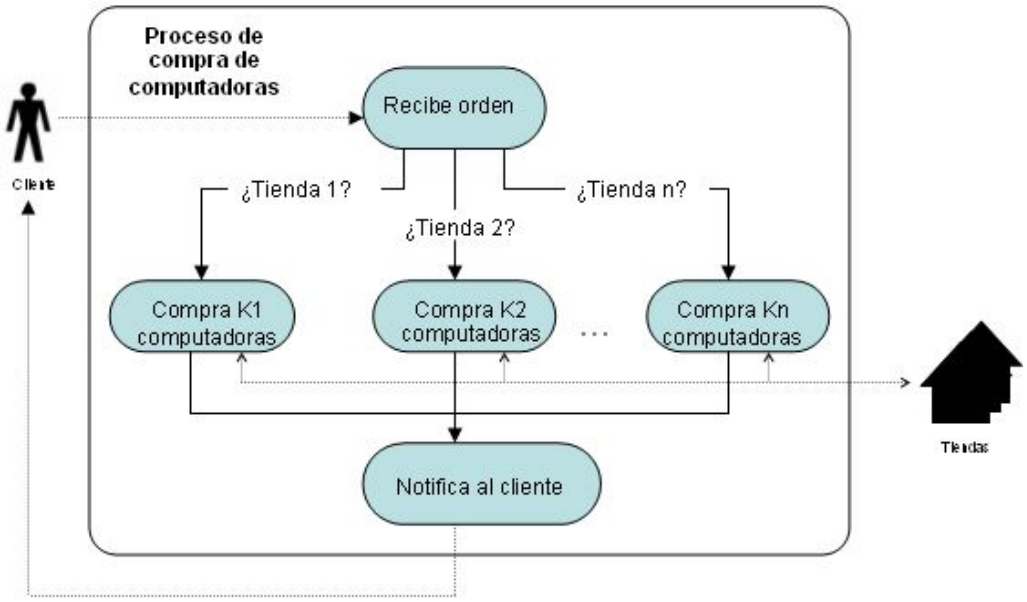

Figura 3.10: Envío de solicitudes de compra a las diferentes tiendas seleccionadas por el cliente.

En la Figura 3.10 se muestra cómo el actor *cliente* solicita la compra de computadoras a un *workflow* que se construye agregando dinámicamente los socios comerciales. La orden es recibida y procede a solicitar en paralelo a cada tienda del conjunto S", K_i computadoras; donde i corresponde a la i-ésima tienda del conjunto. Entonces se espera por todas las notificaciones de confirmación y se envían nuevamente al *cliente*.

Hasta ahora se ha mostrado solamente un ejemplo que ilustra el uso de nuestra solución basada en plantillas. Sin embargo, una amplia variedad de otros *workflows* organizacionales que involucran criterios de optimización han sido desarrollados y probados como *El tiempo de entrega más corto* y *La compra de una lista de artículos en diferentes tiendas*.

A continuación se presenta la interfaz de usuario que el cliente debe usar para realizar la ejecución del *workflow*.

3.4. Interfaz de usuario del portal

En esta sección se presenta la interfaz gráfica de usuario mediante la cual un cliente puede poner en ejecución un *workflow* de compra de artículos. La interfaz se basa en las especificaciones propuestas por RosettaNet para la compra de artículos en Internet. Ahora se explica en qué consiste esta ontología.

3.4.1. Ontologías RosettaNet y UN-SPSC

RosettaNet es un consorcio de compañías dedicadas a la tecnología de la información, componentes electrónicos, manufactura de semiconductores, telecomunicaciones, entre otras. Este consorcio tiene como objetivo crear e implementar estándares de procesos de negocio abiertos y que sean ampliamente aceptados en la industria. Estos estándares sirven fundamentalmente para formar un lenguaje de negocio común en la que los socios de un proceso de negocio se puedan incorporar libremente a una cadena de suministro [30].

RosettaNet ha establecido lo que se conoce como Partner Interface Processes (PIPs) los cuales definen procesos de negocio entre socios comerciales. Los PIPs son interfaces basadas en XML especializadas en la comunicación entre sistemas. Se han clasificado en siete grupos de proceso de negocio capaces de crear una red comercial.

El ejemplo tratado a lo largo de este trabajo concerniente a la compra de una computadora en Internet se ubica en el grupo 3 de *Administración de Ordenes*. Este grupo ayuda en el área de administración de órdenes de los negocios a partir del precio y cantidad de un producto.

El grupo está subdividido en segmentos dentro de los que destaca el segmento 3A. Este

segmento especifica la entrada para obtener la cantidad y orden de un producto. Permite el intercambio de información sobre el precio y disponibilidad, cantidad, órdenes de compra y estado de una orden entre los socios así como el envío de órdenes.

Finalmente, el segmento 3A está conformado por varios PIPs. Los que resultan de interés para nuestro ejemplo son el 3A1 y el 3A2. El primero es conocido como *Petición de Cantidad* y, como su nombre lo indica, permite a un comprador solicitar a un proveedor la cantidad de productos disponibles y a un proveedor regresar en respuesta la cantidad. Una vez que el comprador ha establecido el proveedor con el que desea hacer la petición de compra, la única información que es necesario que envíe es el código del producto. Mientras que el segundo es conocido como *Petición de Precio y Disponibilidad*. Su objetivo es el de proveer un proceso automatizado rápido para los socios comerciales que les permita solicitar y proveer el precio de un producto y su disponibilidad.

Por otro lado, el United Nations Standard Products and Services Code (UN-SPSC) es una ontología promovida por las Naciones Unidas que facilita la clasificación de productos y servicios a través de estándares globales y abiertos.

En nuestro ejemplo, RosettaNet ha permitido la clasificación de los servicios provistos en el portal. Mientras que UN-SPSC se usa para clasificar los productos vendidos en el portal. Así pues, un servicio de *Petición de Precio y Disponibilidad* se identifica como el PIP 3A2 y el producto computadora sigue la ontología de UN-SPSC con el número 431718.

3.4.2. Interfaces de usuario

La interfaz de usuario con la que el cliente interactúa ha sido desarrollada con base en las especificaciones de RosettaNet. Primeramente, el cliente debe seleccionar una de las

ontologías (UN-SPSC, Amazon o STD). En el caso concreto del ejemplo, selecciona la opción UN-SPSC. Después debe seleccionar un producto dentro de una amplia variedad. Supongamos que escoge computers. En la Figura 3.11 se muestra la interfaz a través de la cual el cliente puede hacer las selecciones anteriores.

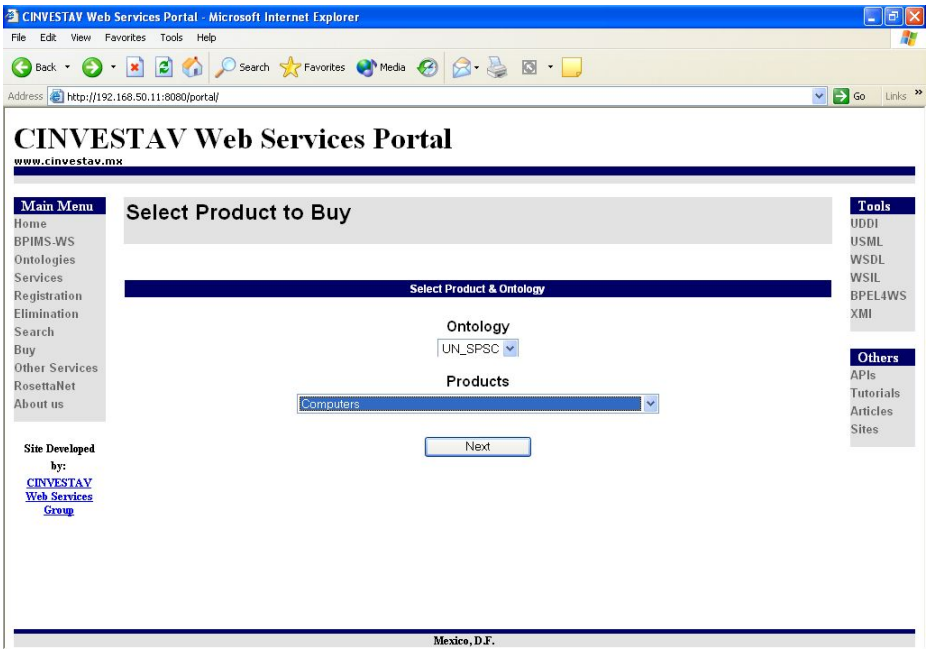

Figura 3.11: Ontologías mostradas al cliente para que realice su compra.

Posteriormente, el cliente selecciona el servicio que le interesa. Puede escoger alguno de los diferentes servicios como *Todos los proveedores*, *precio más bajo*, *precio más bajo y cantidad mínima* o *mejor tiempo de entrega*. El primero muestra todos los proveedores que venden un producto en particular sin ningún tipo de restricciones. El segundo muestra el o los proveedores que ofrezcan el precio más bajo. El tercero obtiene los proveedores que tengan el producto con el menor precio y que tengan como mínimo la cantidad requerida por el cliente. Finalmente, el cuarto muestra el o los proveedores que se comprometan a entregar un producto en el menor tiempo posible. La Figura 3.12 muestra la interfaz que permite la selección de alguno de los servicios.

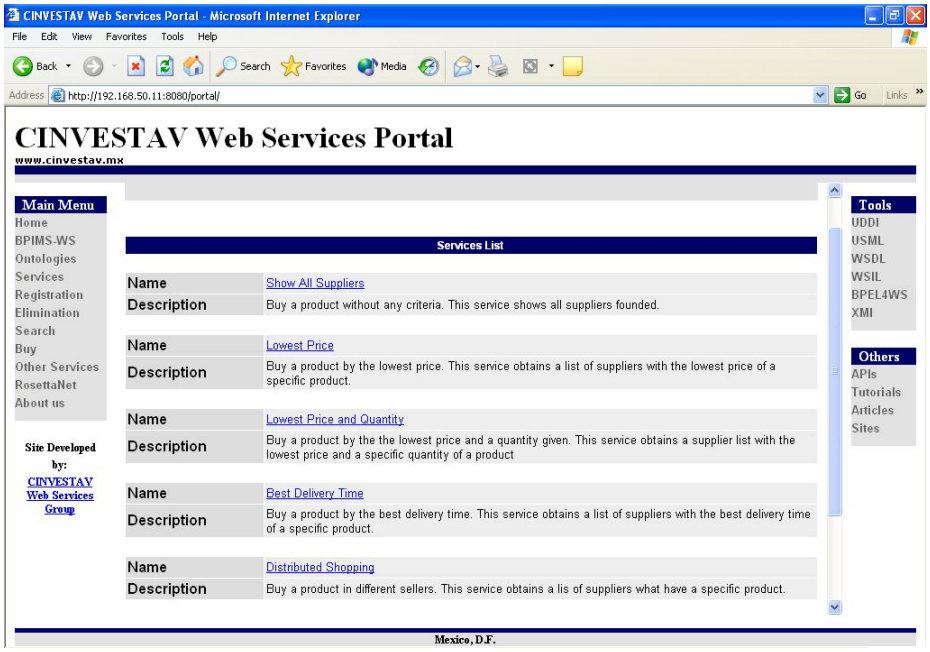

Figura 3.12: Servicios ofrecidos por el portal para la compra de un producto.

Una vez que se ha seleccionado el servicio, el portal envía al servicio orquestado *Precio Más Bajo y cantidad mínima*, el código, ontología y cantidad del producto. Este último procede a ejecutar el *workflow* y responde enviándole la lista solicitada. Entonces se muestran los resultados de la petición. Supongamos que el cliente desea una lista de los proveedores que ofrezcan el menor precio. El resultado de la petición se muestra en la Figura 3.13. Finalmente, el cliente selecciona alguna de las tiendas donde desea hacer la compra y una solicitud de compra es enviada directamente a alguna de las tiendas.

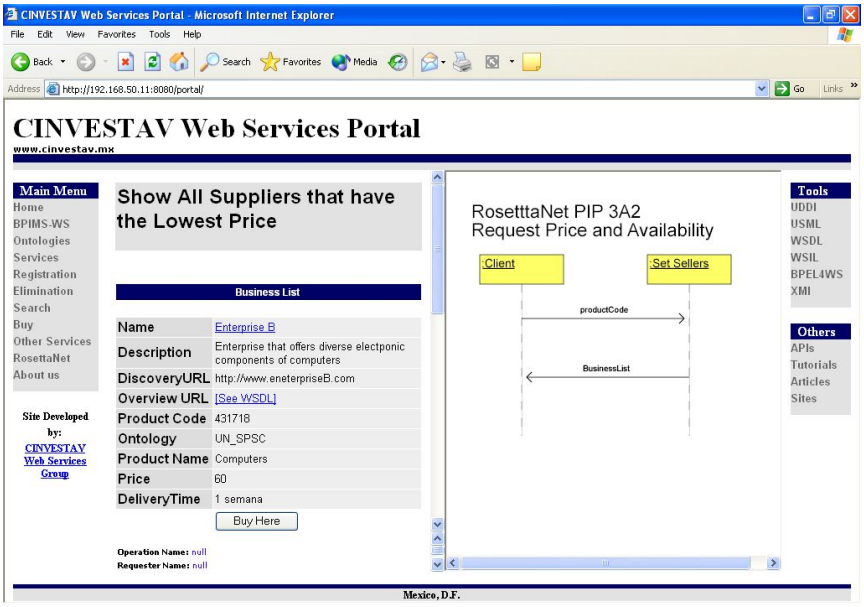

Figura 3.13: Lista de proveedores que ofrecen el precio más bajo de un producto.

3.5. Conclusiones

En este capítulo se mostró un caso de estudio que evidencía la necesidad de contar con un sistema capaz de agregar dinámicamente socios comerciales a procesos de negocio. En la práctica, este tipo de problemas es común; y su solución es de gran utilidad como se muestra en el caso de estudio presentado en este capítulo . Se ejemplificó con un caso particular de compra de un producto en Internet dado el criterio de compra del precio más bajo.

También se propuso una solución consistente de una serie de pasos que permiten agregar dinámicamente los socios al proceso de negocio. Estos pasos son:

- definir un proceso genérico de negocio que abstraiga las prácticas comúnes dentro

de una organización.

- construir un repositorio de plantillas (procesos genéricos) y recuperarlas en tiempo de ejecución.
- concretizar los *workflows* con la información proveniente de los WSDLs de los socios.
- ejecutar el *workflow* en un motor BPEL4WS e interactuar con él, enviándole información y recuperando la respuesta.

El cliente interactúa únicamente con el APN a través de interfaces de usuario generados dinámicamente con JSPs. El APN delega a la máquina BPEL4WS de Oracle la responsabilidad de interactuar con las tiendas o el Intermediario como se verá en el próximo capítulo que muestra detalladamente el deseño del APN.

Capítulo 4

Diseño del Administrador de Procesos de Negocio

Este capítulo tiene como objetivo presentar las perspectivas organizacionales, de datos y funcionales del APN. Estas perspectivas muestran características importantes acerca de su diseño. Asimismo, se muestran los diagramas de secuencia en UML que permiten mostrar las diferentes interacciones que se pueden tener con el APN.

4.1. Perspectiva organizacional

La perspectiva organizacional hace posible la modelación de los actores principales responsables de llevar a cabo las tareas dentro de un *workflow* [32]. En esta perspectiva se definen los objetos organizacionales como los usuarios y sus roles. Los objetos organizacionales pueden clasificarse como agentes, los cuales son entidades capaces de realizar tareas específicas como lo hacen los seres humanos, las máquinas o programas de cómputo.

En el caso de la ejecución del *workflow Precio más bajo y cantidad mínima*, sus participantes consisten de un usuario llamado *Client* y tres servicios Web llamados *partner0LT*, *partner1LT* y *partner2LT* correspondientes a *get_PriceandDeliveryTime*, *get_ProviderURL−Buy* y *get_ProviderQuantity* respectivamente. Esto se puede ver en la Figura 4.1. El cliente puede considerarse como un agente humano que interactúa con el *workflow* a través del portal. Mientras que los servicios Web se consideran como agentes de *software* independientes.

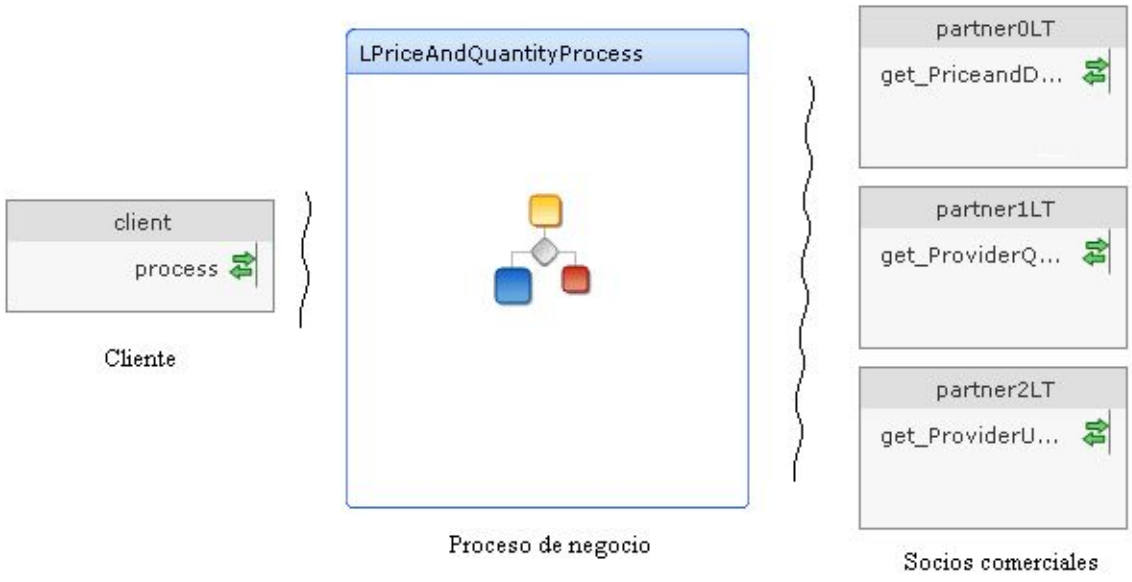

Figura 4.1: Participantes en el proceso de Precio más bajo y cantidad mínima.

BPEL4WS permite la declaración de los participantes en un proceso a través de los *PartnerLinks*. A cada *PartnerLink* se le asocia un nombre, un *PartnerLinkType* y un *role*. Estos tres atributos son generados automáticamente para cada uno de los participantes en el *workflow* del *Precio más bajo y cantidad mínima*. El nombre del *partnerLink* se construye a partir del número de servicio, esto es, el servicio Web *n* tiene un *partnerLink* llamado *partnernLT*. El tipo de este *partnerLink* se compone de un espacio de nombres y del *partnerLinkType* correspondiente al servicio Web. El *partnerLinkType* tiene entonces la

forma *pnN:partnerNPLT*, donde N es el número asociado con el servicio Web. Finalmente, el *role* de cada uno de los participantes se determina agregando un atributo llamado *partnerRole* a quien se le asocia el valor *partnerNProvider*.

Además, cada uno de los servicios Web involucrados en el *workflow*, incluyendo al servicio Web orquestado, proveen puertos por donde se efectúa la comunicación con ellos. Estos puertos se descubren en tiempo de ejecución y se agregan dinámicamente cuando se hace la invocación del servicio. Los puertos correspondientes al cliente y a los tres servicios Web se muestran en la Figura 4.1 y se llaman process, $get\_PriceandDeliveryTime$, $get\_Provider-Quantity$ y $get\_ProviderURLBuy$ respectivamente.

4.2. Perspectiva de datos

En esta perspectiva se muestran los datos manejados en el *workflow* y la información que el APN administra para poder concretizar y ejecutar las plantillas.

4.2.1. Variables locales

El *workflow* posee variables que marcan su estado o permiten el intercambio de información con otros agentes. En el caso del *workflow LPiceAndQuantity*, se han declarado variables que pertenecen a la plantilla y que permiten la implementación del algoritmo para la obtención del precio más bajo y cuyos tipos son primitivos (enteros, cadenas, etc.). También forman parte de la plantilla las variables de entrada y salida del *workflow* que permiten la comunicación con el cliente y cuyo tipo es *LPriceAndQuantityRequestMessage* y *LPriceAndQuantityResponseMessage* respectivamente. Estos tipos han sido declarados en el documento WSDL del servicio Web orquestado que define su interfaz.

Sin embargo, también se encuentran variables que son construidas dinámicamente para el intercambio de información con los servicios Web externos. A continuación se explica el procedimiento que se sigue para su construcción.

4.2.2. Mensajes de entrada y salida del *workflow*

Se ha descrito en las secciones anteriores la forma en que BPEL4WS recibe y envía la información proveniente y dirigida al portal; esto lo hace a través de las operaciones *receive* y *reply* respectivamente. Todas las plantillas usan un *receive* que marca el inicio de la ejecución del *workflow* y un *reply* que marca su final. La operación *receive* requiere de una variable IN (de entrada y sólo lectura) y *reply* de una variable OUT (de salida y sólo escritura). Por ello, estas variables están presentes en todas las plantillas.

Al interior del *workflow* se establece contacto con servicios Web externos. El intercambio de información entre el *workflow* y ellos queda determinada por dos variables que son agregadas dinámicamente a la sentencia *invoke* con la que se invoca un servicio Web participante en el *workflow*. En este trabajo sólo se han trabajado con envíos síncronos de información; una variable de entrada y otra de salida son pues necesarias. Su nombre está relacionado con la función que realizan y con el servicio Web con el que se comunican. Así, una invocación al servicio Web *n*, implica la generación de una variable de entrada *partnernRequest* y otra de salida llamada *partnernResponse*. Un problema fundamental es hacer corresponder el *messageType* de cada una de las variables con los messages de entrada y salida provistos en el WSDL del servicio Web externo. La solución que se ha propuesto es analizar el documento WSDL y a través de WSIF [33] recuperar la operación que provee y entonces capturar los mensajes de entrada y salida correspondientes a la operación. WSIF es una API en Java promovida por Apache Software Foundation ampliamente usada para la invocación de servicios Web. El uso que se le da básicamen-

te en este trabajo es la de inspeccionar el WSDL de los servicios Web socios como se verá detalladamente en el capítulo 5.

4.2.3. Diseño del repositorio de plantillas en BPEL4WS

El repositorio de plantillas BPEL4WS se diseñó con el fin de ubicar eficazmente las plantillas que abstraen un proceso de negocio específico. La Figura 4.2 muestra el diseño de dicho repositorio que puede ser mapeado a tablas de una base de datos.

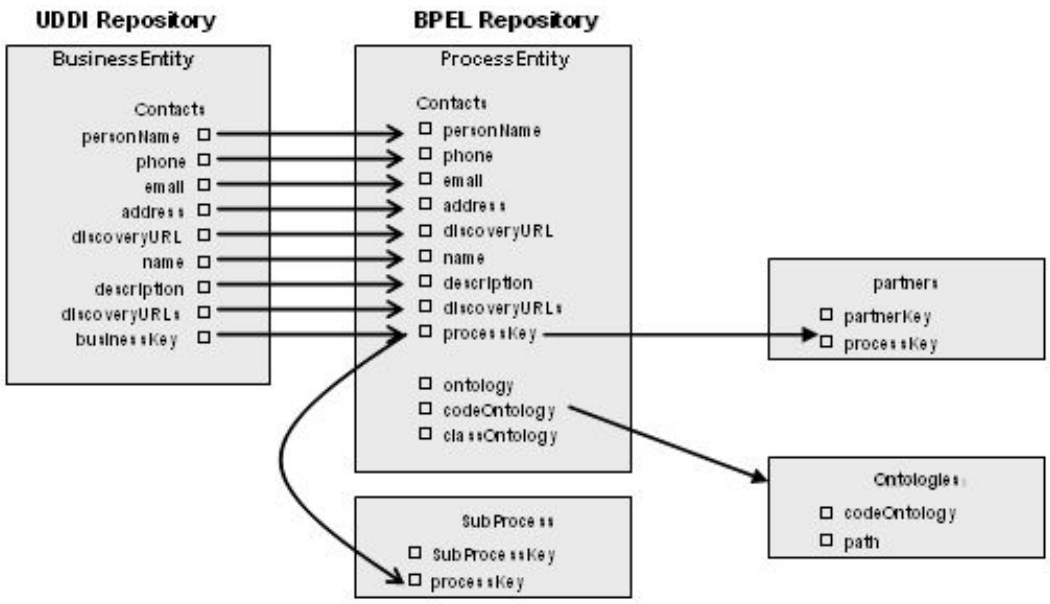

Figura 4.2: Esquema del repositorio BPEL4WS.

Se puede ver claramente en la figura la similitud que se tiene con el repositorio UDDI. La información equivalente a las páginas blancas (nombre, teléfono, e-mail, dirección, etc.) sirve para identificar a los creadores del proceso de negocio. Así pues el autor del proceso de negocio genérico puede ingresar sus datos en esta sección.

Sin embargo, existen algunas extensiones y modificaciones que se han hecho. Se ha cambiado la clave del negocio (*businessKey*) por la clave del proceso (*processKey*). Esta clave sirve para identificar de manera única un proceso (proceso de compra en este caso). También se han agregado tres tablas correspondientes a los socios, ontologías y subprocesos de un proceso de negocio.

La tabla de socios se basa en la idea de que un proceso cuenta con uno o más socios que deben agregarse en tiempo de diseño. Es necesario precisar que en el caso de la compra distribuida, debido a que no se conocen de antemano los socios, el proceso no se relaciona con ningún socio en tiempo de diseño y por tanto el código de este proceso no se encuentra en la tabla de socios. En el caso de la búsqueda de productos se sabe que los únicos socios son los que provee el intermediario y por ello sí existe un proceso que se relaciona con un conjunto no vacío de socios en la tabla de socios.

La tabla de ontologías tiene la finalidad de establecer claramente el tipo de servicio que se está ofreciendo de acuerdo a las taxonomías de RosettaNet. Por ejemplo, la compra de un producto corresponde a los PIPs 3A2 de RosettaNet. Esta taxonomía especifica que el precio y tiempo de entrega de un producto deben ser determinados únicamente a partir de su código. En cuanto a los campos *ontology*, *codeOntology* y *classOntology* que se encuentran en la tabla de procesos, permiten relacionar un proceso con una ontología que se encuentra dentro de la tabla de ontologías.

Finalmente, la tabla de subprocesos permite relacionar un proceso con varios subprocesos necesarios para la correcta ejecución del *workflow* a partir del *processKey* que se encuentra en la tabla de procesos. En este trabajo no se utiliza este campo aunque en trabajos futuros resultará de gran utilidad para incorporar *workflows* a otros *workflows*.

4.2.4. Estructura de directorios

La estructura de los directorios donde se encuentran las plantillas BPEL4WS y WSDL, así como sus archivos de configuración se muestran en la Figura 4.3. El directorio base es el *BPELRepository*. Al interior de este directorio se encuentra un directorio llamado *Templates* donde se encuentran a su vez los diferentes *workflows* correspondientes a las ontologías de RosettaNet: 3A2-1, 3A2-2, etc.

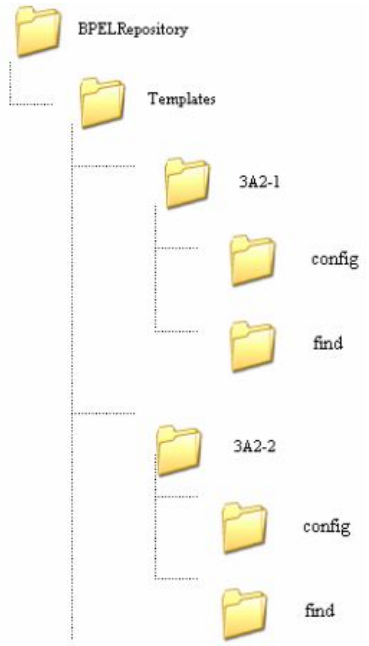

Figura 4.3: Estructura del directorio del repositorio de plantilla BPEL4WS.

Cada uno de los directorios correspondientes a las ontologías contiene los subdirectorios de configuración y de las plantillas del BPEL4WS y WSDL (*config* y *find* respectivamente). A continuación se explica el contenido de cada uno de estos subdirectorios.

4.2.4.1. Directorio de configuración y del *workflow*

El directorio de configuración contiene las plantillas de los documentos *.PROJECT*, *build.xml* y *bpel.xml*. En el primero se establece esencialmente el nombre del proyecto y comandos para compilar el documento BPEL4WS. En el segundo se declara el nombre del proyecto que se va a compilar y la ruta donde se encuentra el documento *bpel.xml*. En el último archivo se define un identificador de proceso que es el mismo que el nombre del proyecto, el nombre del documento BPEL4WS fuente y del WSDL los cuales están formados por el nombre del proyecto concatenado con la extensión *.bpel* y *.wsdl* respectivamente. Asimismo, se definen las direcciones de los documentos WSDL participantes en el *workflow*.

El directorio del *workflow* contiene únicamente las plantillas de los documentos BPEL4WS y WSDL correspondientes a las ontologías.

4.2.4.2. Directorio de ejecución

Una vez que se han concretizado las diferentes plantillas, se construye un directorio temporal dentro del directorio de la ontología llamado *ServiceK*, donde K es el número de la sesión atendida por el portal. Este directorio está construido de la forma en que se muestra en la Figura 4.4.

Se puede apreciar que los archivos de configuración y del *workflow* se ubican dentro del directorio *ServiceK* una vez que han sido concretizados. Además, se crea un sub directorio llamado services donde se ponen los diferentes *wrappers* correspondientes a los participantes en el *workflow*. Cada uno de estos *wrappers* contiene la dirección del participante que puede encontrarse en el propio servidor o en una dirección remota en Internet. También

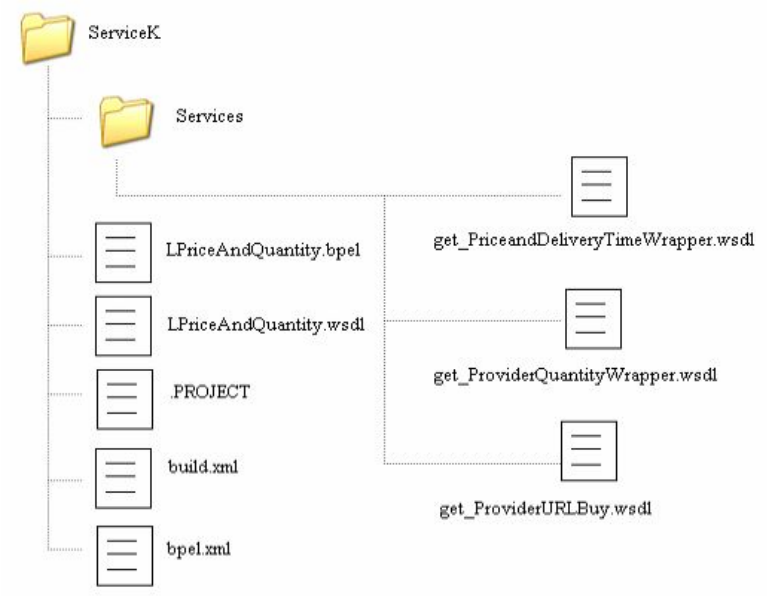

Figura 4.4: Estructura del directorio de ejecución temporal.

se define el *PartnerLinkType*, el *role* y el *portType* con los que se identifica al servicio participante como se explicó en el capítulo anterior.

4.3. Perspectiva funcional

En esta sección se muestran las funciones que se cumplen durante la ejecución del *workflow* y las efectuadas por el APN.

4.3.1. El *workflow*

Durante la ejecución del *workflow* se hacen invocaciones repetidas a los diferentes servicios Web. Esto se logra con la construcción dinámica de sentencias *invoke* que tienen la forma

de la Figura 4.5.

```
<invoke name="invokeService"
        inputVariable="partnerNRequest"
        outputVariable="partnerNResponse"
        portType="pnN:WebServiceNamePortType"
        operation="operationName"
        partnerLink="partnerNLT"/>
```

Figura 4.5: Estructura de una sentencia invoke.

Esta sentencia permite invocar la operación *operationName* ofrecida en el puerto *WebServiceNamePortType* por el servicio Web *N*. La invocación de dicha operación es síncrona; la variable de entrada es *partnerNRequest* y la variable de resultado es *PartnerNResponse*.

4.3.2. El APN

En esta sección se muestra la forma en que procede el APN ante una solicitud de un cliente. En el Cuadro 4.1 se presenta el caso de uso *Compra de un producto a través del portal* usando el APN.

Esquemáticamente se muestra el proceso completo en la Figura 4.6. En el paso **1**, el cliente solicita un servicio a través del portal. A partir de la información proporcionada por el cliente, el portal procede a encontrar los socios y las rutas de las plantillas en la base de datos correspondiente al repositorio de plantillas. Esta última información se envía al APN para que proceda a recuperar todas las plantillas en el paso **2**. En el paso **3**, las plantillas son recuperadas y en el paso **4** son concretizadas con la información obtenida por el portal. Entonces, en el paso **5**, las plantillas debidamente llenadas se envían para su ejecución a la máquina BPEL4WS. El portal formula entonces una invocación usando JAX-RPC al *workflow* que se encuentra ejecutándose en la máquina BPEL4WS. El *workflow* comienza

Cuadro 4.1: Caso de uso *Compra de un producto a través del portal*

Cliente	**APN**
1 Escoge una ontología, Selecciona un servicio y Selecciona un producto.	
	2 Busca en la BD las plantillas
	3 Recupera las plantillas
	4 Se concretizan las plantillas de proceso y configuración.
	5 Se instala el proceso en el APN.
	6 Se ejecuta el proceso pasándole los valores ingresados por el cliente.
	7 Encuentra los servicios Web candidatos.
8 Se obtiene la respuesta.	
	9 Se desinstala el servicio

su ejecución invocando los servicios del Intermediario en el paso **6**. El Intermediario busca los servicios Web de los socios comerciales en el paso **7** que se encuentran en un repositorio UDDI privado que en esencia es una base de datos donde se registran los servicios Web que ofrecen cierto tipo de servicios. Finalmente, en el paso **8** regresa los resultados al *Portal* que a su vez los envía al cliente. Al completarse el paso **8**, el cliente procede a escoger las tiendas con las que desea comprar sus productos. El APN crea el *workflow* de *Solicitud de Compra* y lo ejecuta en la máquina BEPL4WS. Durante la ejecución, el *workflow* realiza el paso **9** donde invoca directamente a los servicios Web de las diferentes tiendas y obtiene los resultados de la invocación como la confirmación de compra.

4.4. Comportamiento

En esta sección se muestra primeramente la interacción que el cliente tiene con el portal para comenzar con la ejecución del *workflow*. Posteriormente, se muestra la interacción que el *workflow* tiene con el cliente y los servicios Web que invoca.

4.4.1. Interacción con el APN

Desde la perspectiva del cliente, se pueden identificar tres actores que intervienen en el cumplimiento de su solicitud. Estos tres actores se pueden ver en la Figura 4.7.

Se puede ver en la Figura 4.7 que el actor *Usuario* inicia el envío de mensajes. Primeramente, envía el código, la ontología y la cantidad solicitada del producto al actor *Portal*. A su vez, éste solicita al APN la construcción y ejecución de un *workflow*. El APN responde concretizando las plantillas y ejecutándolas en la máquina BPEL4WS. Al recibir los resultados, al APN los regresa al *Portal* que los envía al *Usuario* que inició la solicitud.

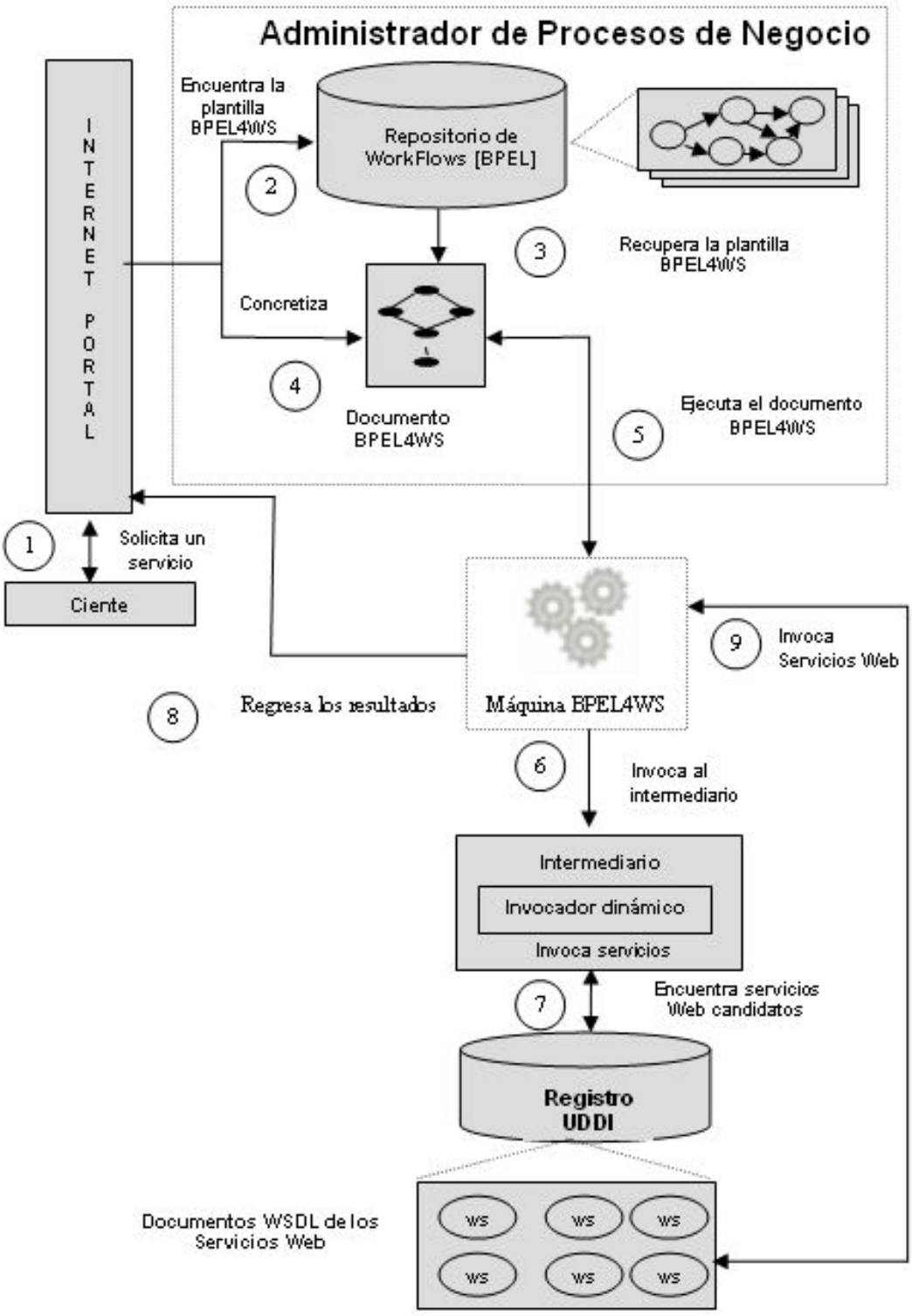

Figura 4.6: Pasos para la composición dinámica de procesos de negocio.

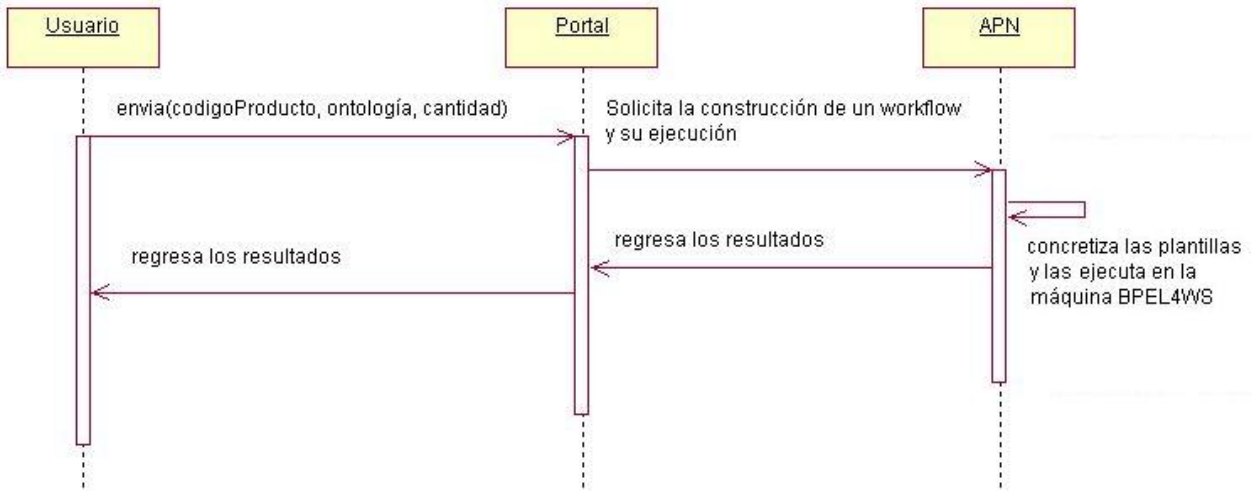

Figura 4.7: Diagrama de secuencia de *Solicitud de lista de productos al menor precio y cantidad mínima.*

4.4.2. Interacción con el *workflow*

Desde la perspectiva del *workflow*, se tienen cinco actores que intervienen en su ejecución. La Figura 4.8 muestra el diagrama de secuencia de *La ejecución del* workflow *Precio más bajo y mínima cantidad.*

Como se puede ver en el diagrama de la Figura 4.8, el actor *Cliente* inicia la ejecución del *workflow*. Envía a este último el código, ontología y cantidad del producto requerido. El *workflow* procede a enviar al servicio Web *get_PriceandDeliveryTime* su código y ontología. Este responde enviando de regreso el *businessKey*, precio y tiempo de entrega del producto. Entonces el *workflow* envía al servicio Web *get_ProviderQuantity*, el *businessKey* que obtuvo del servicio Web anterior, el código y la ontología. Este responde pasándole la cantidad, el nombre de la empresa que lo provee, su descripción, el *discoveryURL* y su nombre. Finalmente, el *workflow* solicita al servicio Web *get_ProviderURLBuy* la dirección en Internet donde puede hacer la compra del producto pasándole el *business-*

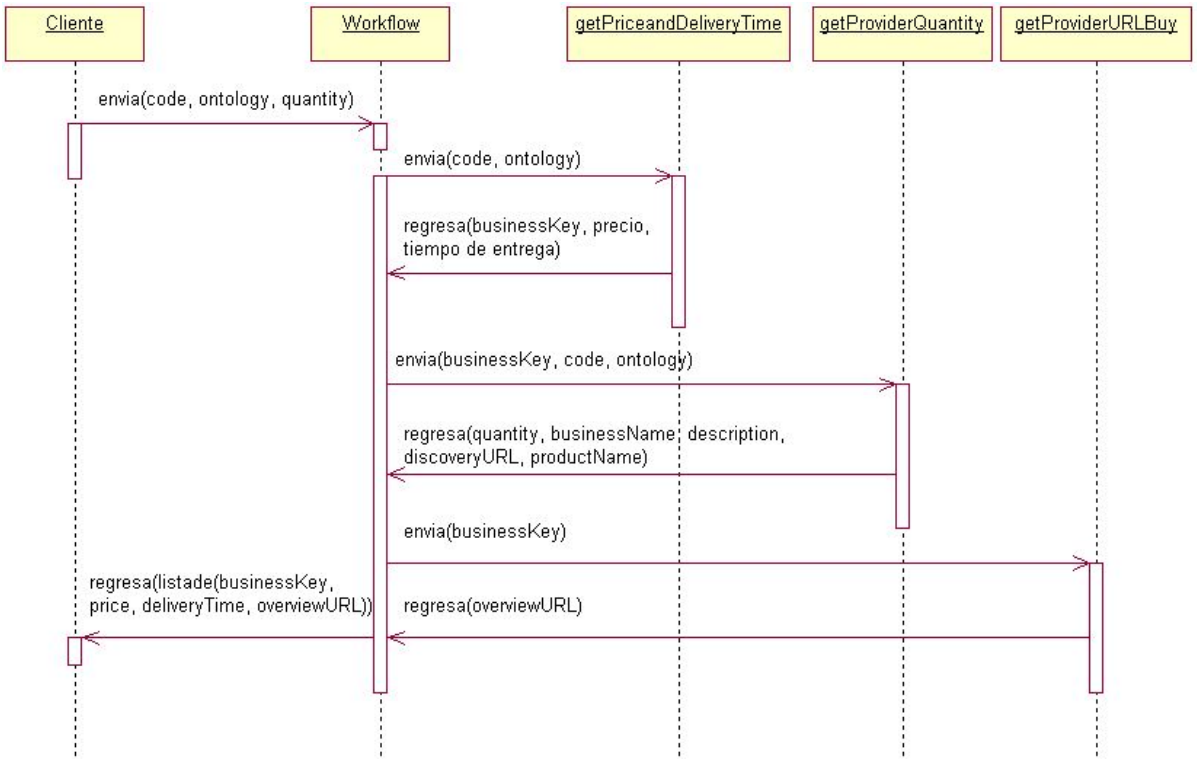

Figura 4.8: Diagrama de secuencia de *La ejecución del* workflow *Precio más bajo y mínima cantidad.*

Key de la empresa. El *workflow* regresa entonces al cliente una lista de las empresas que proveen el precio más bajo, y que tienen como mínimo la cantidad solicitada por el cliente.

4.5. Conclusiones

En este capítulo se mostraron las perspectivas oraganizacionales, de datos y funcionales del APN. En la perspectiva organizacional se revelan los diferentes actores participantes en un proceso de negocio y la forma en que se declaran en BPEL4WS.

En la perspectiva de datos se muestra la forma en que se declara la información que fluye al interior del *workflow* y la que se envía hacia los participantes y se recibe de éstos. También se muestra la forma en que el APN administra la información necesaria para recuperar las plantillas. El APN interactúa con una base de datos que constituye el repositorio de las plantillas. Este repositorio contiene la información necesaria para identificar la ubicación de dichas plantillas y eventualmente los socios que pueden ser identificados en tiempo de diseño. Además, se muestra la estructura de directorios del repositorio de plantillas organizado conforme a la ontología de RosettaNet.

En cuanto a la perspectiva funcional, ésta muestra la forma en que el *workflow* se comunica con los servicios Web participantes que, desde la perspectiva del *workflow*, encapsulan funciones específicas que se incorporan a la lógica del negocio.

Por último, se explica la interacción entre los diferentes actores que intervienen en la solicitud de la compra de un producto al precio más bajo a través de diagramas de secuencia de UML. En el capítulo siguiente se muestra la implementación del APN.

Capítulo 5

Implementación

En la actualidad existe una gran cantidad de tecnología asociada con los servicios Web que facilitan su uso. Muchas son las empresas desarrolladoras de *software* que dan soporte al desarrollo de servicios Web como Microsoft a través de su plataforma .Net, Sun Microsystems con Java e IBM con su Web Sphere por mencionar sólo algunos. En esta tesis se ha trabajado ampliamente con diversas clases escritas en Java debido a que pueden ser libremente usadas. Además, tienen el soporte de organizaciones como el Apache Software Foundation e IBM.

En la primera parte de este capítulo se muestran las clases que constituyen al APN así como su jerarquización desde la perspectiva de la programación orientada a objetos. Posteriormente, se muestran las clases usadas para las conexiones a las bases de datos, la inspección de los servicios Web, la manipulación de las plantillas y de la información que maneja el APN en XML y la comunicación con el *workflow*.

5.1. Clases de los archivos de configuración

La máquina BPEL4WS usada a lo largo de este trabajo para desplegar los servicios Web orquestados es la provista por Oracle. Para desplegar un servicio Web orquestado es necesario generar archivos de configuración y de ejecución. Los primeros permiten establecer el nombre, socios e información adicional del servicio Web. Los segundos tienen que ver con el documento BPEL4WS y el WSDL. Se han construido clases para cada uno de estos archivos que generan los documentos concretizados a partir de las plantillas. A continuación se describen cada una de estas clases.

5.1.1. La clase Project

Esta clase es sumamente simple y pequeña. Su estructura se muestra en la Figura 5.1. Como se puede apreciar consiste de un método llamado **createFile()** que se encarga únicamente de recuperar del directorio de configuración la plantilla del documento *.PROJECT* y de agregarle el nombre que le ha sido asignado al proyecto. Entonces copia el documento así modificado en el directorio *ServiceK*.

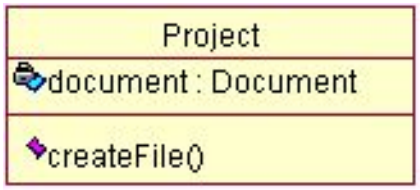

Figura 5.1: Clase Project.

5.1.2. La clase BuildXml

Esta clase es muy similar a la anterior. Está constituida por un método llamado **createBuildXml()** que recupera la plantilla del documento *Build.xml*, le agrega el nombre del proyecto y la sitúa en el directorio *ServiceK*. Su estructura se muestra en la Figura 5.2.

Figura 5.2: Clase BuildXml.

5.1.3. La clase BPELXml

Al igual que las clases anteriores, ésta se compone por un solo método llamado **createBpelXml()**. Primeramente, recupera el documento *bpel.xml* y le agrega los atributos *id*, *src* y *wsdlLocation* construidos a partir del nombre del proyecto. Posteriormente, agrega los *PartnerLinks* con la ubicación de los *wrappers*. El nombre del *PartnerLink* se compone por el número asociado con el servicio Web participante (por ej. *ParnerkLT* donde k es el número del servicio Web). Finalmente, posiciona el documento construido en el directorio *ServiceK*. Su estructura se muestra en la Figura 5.3.

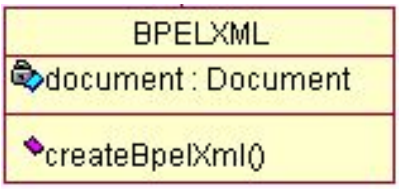

Figura 5.3: Clase BPELXml.

5.2. Diagrama de clases de los archivos de ejecución

5.2.1. La clase Partners

Esta clase tiene varios métodos a partir de los cuales se construye el directorio *services* con los *wrappers* correspondientes a los servicios Web participantes en el *workflow*. El método **setPartners()** recibe como parámetros las direcciones de dichos servicios Web, y crea un *wrapper* para cada uno de ellos. El nombre del *wrapper* está constituido por el nombre del servicio Web concatenado con la palabra *wrapper*.

Por otro lado, el contenido del *wrapper* se crea también dinámicamente. Se recuperan los espacios de nombre que se encuentran en los WSDL de los socios comerciales y se agregan en un elemento definitions del *wrapper*. También se agrega un elemento *import* donde se especifica la dirección del servicio Web. Finalmente se agrega un elemento *PartnerLikType* cuyo nombre es de la forma *parnerkPLT*, y dentro de este elemento se agrega un elemento *role* cuyo nombre es de la forma *partnerkProvider*, donde k representa el nombre del servicio Web. Finalmente, dentro de este último elemento se agrega el *portType* provisto por el participante junto con su espacio de nombres.

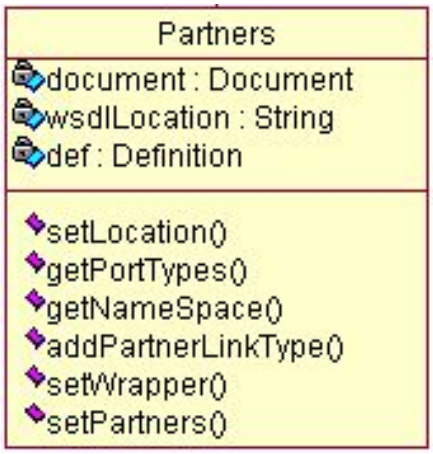

Figura 5.4: Clase Partners.

5.2.2. La clase WSDLFile

Esta clase tiene como objetivo pasar el documento WSDL que define la interfaz disponible por el servicio Web orquestado al directorio *serviceK*.

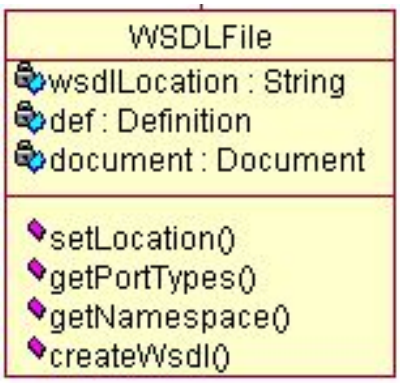

Figura 5.5: Clase WSDLFile.

5.2.3. La clase BPELFile

La clase **BPELFile** es quizás, la más importante de todas. Su función es la de proveer los mecanismos necesarios para que las clases **BPELFileLP**, **BPELFileLPQ**, etc. puedan concretizar las plantillas correspondientes a las diversas ontologías. En la Figura 5.6 se muestra la jerarquía de clases.

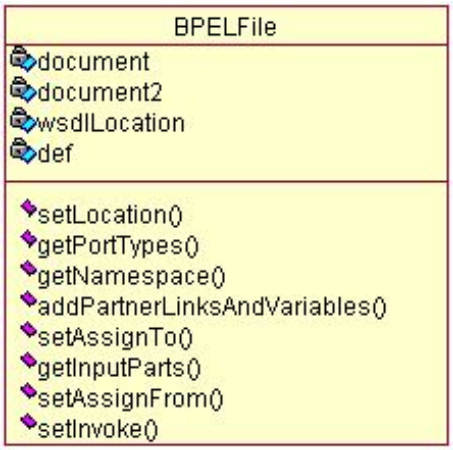

Figura 5.6: Clase BPELFile.

Como se puede apreciar en la figura anterior, las clases **BPELFileLP** y **BPELFileLPQ** derivan a la clase **BPELFile** e implementan la interfaz **BusinessLogic**. La razón por la cual se ha creado esta jerarquía es porque cada plantilla BPEL4WS se llena de una manera distinta. Por lo que cada clase que concretiza la plantilla implementa su propia lógica de negocio. Sin embargo, cada una de estas clases hacen uso de los mismos mecanismos para concretizar las platillas, por lo que heredan de la clase **BPELFile** los mecanismos.

5.2.4. Clase APN

Esta clase posee un método que permite crear y ejecutar del proceso de negocio llamado **createAndExecuteProcess()**. Como se puede ver en la Figura 5.7, instancias de las clases **Project**, **BuildXml**, **Partners**, **BPELXML** y **WSDLFile** se agregan al APN y se usan para concretizar las respectivas plantillas.

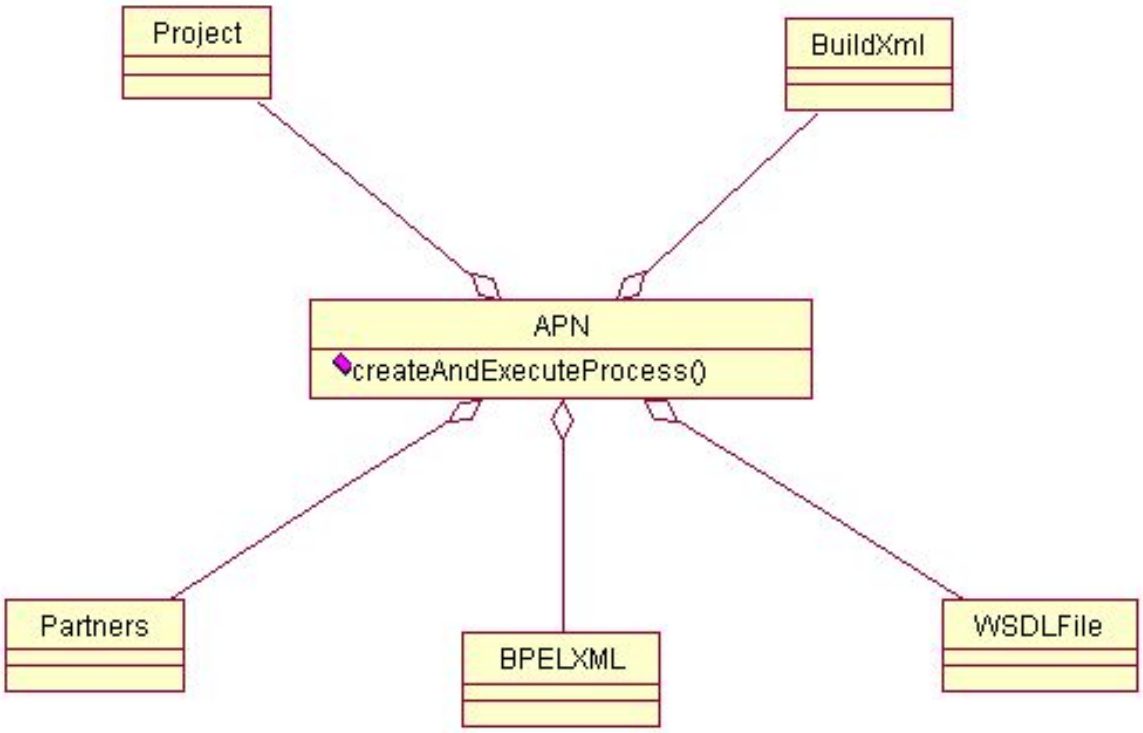

Figura 5.7: Clase APN.

5.3. Diagrama general de clases del APN

El diagrama general de clases del APN se muestra en la Figura 5.8. Como se puede ver, la clase **DocumentUtils** se encuentra en la parte superior de la jerarquía. Básicamente el uso que tiene esta clase es la de crear documentos XML a partir de otros documentos o desde su inicio. Todas las clases anteriormente descritas usan los métodos de esta clase. También existe una clase **XMLDocument** que tiene como objetivo manipular los elementos de un documento XML.

5.4. Conexiones JDBC/ODBC al repositorio en Access

Las conexiones a la base de datos de Access representativa del repositorio de plantillas BPEL4WS se efectúan a través de JDBC/ODBC. Desde el portal se crean las conexiones en la forma tradicional. La única operación realizada sobre la base de datos es de consulta ya que lo único que se desea hacer sobre la base de datos es extraer la ubicación de las plantillas BPEL4WS.

5.5. Inspección de los servicios Web

Una vez ubicado el documento WSDL correspondiente a un servicio Web el APN procede a su análisis. Para realizar dicho análisis, el APN usa el Web Services Invocation Framework (WSIF).

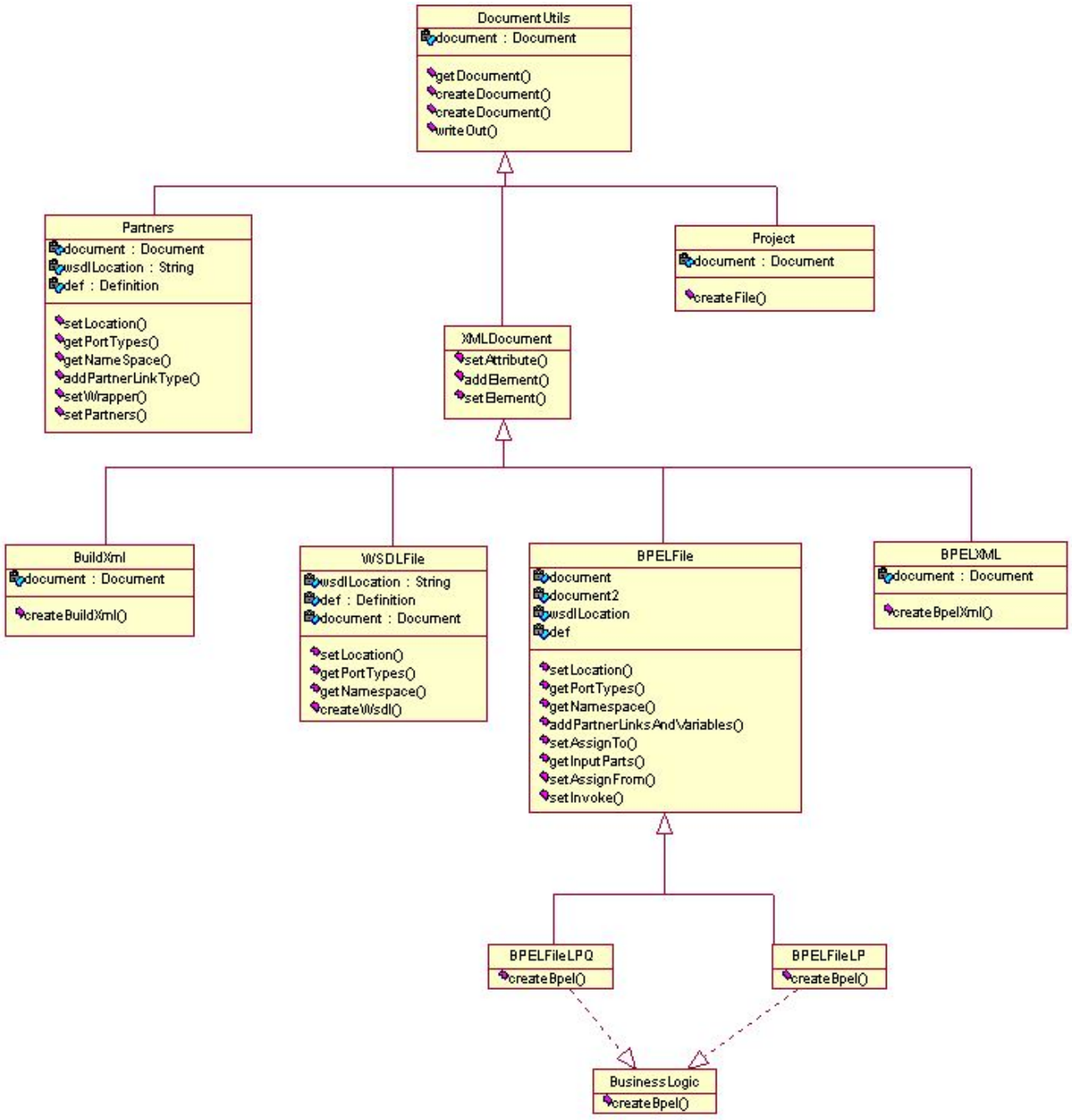

Figura 5.8: Diagrama general de clases.

WSIF es una API en Java para la invocación de los servicios Web, no importando dónde y cómo los servicios se proveen [12]. WSIF contiene algunas utilerías con las cuales se pueden obtener objetos del Web Services Description Language for Java (WSDL4J) [34]. El WSDL4J es una API en Java de WSDL que permite mapear la información encontrada en documentos WSDL a objetos Java que pueden manipularse [31].

Se han creado métodos basados en WSIF, que permiten la inspección de los WSDL de los participantes en el *workflow*. La posterior construcción de los *partnerLinks* se deriva de esta inspección.

Básicamente se usan dos métodos de WSIF. El primero se llama **readWSDL()** y se encuentra dentro de la clase **WSIFUtils**. Este método permite la localización de un documento WSDL y su conversión a objetos Java; el método regresa un objeto **Definition** a partir del cual se pueden obtener los *portTypes* declarados en el WSDL. La operación que hace uso de este método es **setLocation()** que se encuentra dentro de la clase **Partners** y cuyo parámetro es la ubicación del documento WSDL. El otro método que ha sido muy importante es **getAllItems()** el cual recibe un objeto **Definition** y el nombre del elemento que es de interés; en este caso es **PortType**. El método que lo utiliza es **getPortTypes()**. Su resultado es un mapa de objetos **PortType**s de los cuales se obtiene el nombre que debe ser agregado en el documento BPEL4WS y en los *wrappers*.

5.6. Manipulación de las plantillas a través del DOM

En este trabajo se han usado algunas librerías de Apache que permiten manipular documentos XML. En particular, Xalan es una librería muy útil que provee parsers como el **DocumentBuilderFactory** que define una API que permite a las aplicaciones obtener un *parser* que produce árboles de objetos DOM a partir de un documento XML.

DOM es una iniciativa del W3C y define un conjunto de objetos e interfaces para acceder y manipular documentos XML. Representa a los documentos como una jerarquía de objetos **Node**. Estos objetos son de dos tipos: tienen un hijo o son hojas de un árbol.

El **DocumentBuilderFactory** se usa para crear un objeto DOM llamado **Document** que representa al documento XML analizado. A partir de este objeto se pueden añadir y modificar los objetos **Element** que constituyen los nodos del árbol y heredan del objeto **Node** perteneciente al DOM.

Así pues, las plantillas son accedidas a través del **DocumentBuilderFactory** de Xalan y su manipulación (añadir, quitar o modificar los elementos) se realiza con la ayuda de DOM.

5.7. Manipulación de la información a través de XPath

En la Introducción se habló de la forma en que XPath se utiliza para manipular la información intercambiada con los servicios Web. En esta sección se muestran las extensiones a XPath usadas en este trabajo que brindan mayor flexibilidad a esta manipulación.

Básicamente existen dos extensiones a XPath que han sido usadas: la de BPEL4WS y la de Oracle. Ambas extensiones son bastante útiles para manipular cualquier información expresada en XML.

Las funciones provistas por la extensión hecha por PBEL4WS se encuentran en el espacio de nombres *bpws*. La función más usada ha sido **getVariableData()** y tiene la forma:

$bpws : getVariableData(variableName, partName, locationPath) \rightarrow Node$

Esta función extrae cualquier valor contenido en alguna variable BPEL4WS. únicamente se le debe pasar como parámetros el nombre de la variable, el nombre de la parte y la ruta XPath. Su resultado es un nodo.

Las extensiones hechas por Oracle se encuentran en el espacio de nombres

ora =http://schemas.oracle.com/xpath/extension

Las funciones usadas son las mostradas en la Tabla 5.1.

5.8. Comunicación con el *workflow* a través de JAX-RPC

5.8.1. Invocación estática vs invocación dinámica en JAX-RPC

La invocación del *workflow* desde el portal se lleva a cabo usando el Remote Procedure Call (RPC). Como su nombre lo indica, RPC es un mecanismo por medio del cual una aplicación cliente hace llamados a procedimientos remotos pertenecientes a una aplicación servidor. En esta tesis se ha trabajado con una versión de RPC desarrollada por Apache Axis llamada JAX-RPC, la cual provee los medios necesarios para implementar RPC sobre SOAP.

Una aplicación cliente puede realizar la invocación de un procedimiento remoto de tres maneras distintas. La primera es síncrona y se efectúa en modo petición-respuesta. En este caso se invoca el procedimiento remoto y se bloquea la ejecución de la aplicación cliente hasta que obtenga una respuesta de la aplicación servidor. La segunda es en un solo sentido, es decir, la aplicación cliente invoca el procedimiento remoto pero no bloquea

Cuadro 5.1: Funciones de Oracle que permiten la manipulación de variables.

Función	**Descripción**
ora:countNodes(variableName, partName, locationName) → number	Esta función cuenta el número de nodos que se encuentran una variable BPEL4WS. Se le pasa como parámetros el nombre de la variable, el nombre de la parte y la ruta del nodo a contar. Regresa un número.
ora:getNodeValue(aNode) → *string*	Esta función obtiene el valor contenido en un nodo. Se le pasa como parámetro el nodo en cuestión. Regresa el valor en cadena.
ora:mergeChildNodes(element1, element2) → *node*	Esta función mezcla los nodos hijo de dos elementos en un solo nodo. Se le pasa como parámetros los dos elementos. Regresa un nodo.

su ejecución ya que no espera por una respuesta. El tercer tipo de invocación es no bloqueante. En este caso la aplicación cliente realiza la invocación pero no se bloquea su ejecución, aunque posteriormente tiene la posibilidad de pedir la respuesta.

Ahora bien, JAX-RPC posee mecanismos representados en un modelo estático y dos dinámicos a través de los cuales una aplicación cliente invoca un servicio remoto. En el modelo estático se usan *stubs* y herramientas que generan el código de éstos. En uno de los modelos dinámicos se genera un objeto proxy dinámicamente y en el otro se usa el Dynamic Invocation Interface (DII) el cual usa un objeto **Call**.

El modelo usado para la invocación del *worflow* es el estático debido a que posee ventajas relevantes para este trabajo con respecto a los otros dos modelos. Este modelo es limitado porque el *stub* debe conocer la interfaz remota acerca del servicio en tiempo de compilación. En contraste, el DII permite a la aplicación cliente descubrir interfaces remotas dinámicamente e invocar métodos de objetos que implementan estas interfaces. Esto es importante ya que no se requiere saber nada acerca de la interfaz remota en tiempo de compilación y se puede dejar hasta el tiempo de ejecución su descubrimiento. Además DII permite el envío síncrono y en un solo sentido, en tanto que el modelo estático sólo permite el envío síncrono. Sin embargo, un notable y lamentable problema que presenta DII es que no es completamente dinámico en el sentido de que sólo permite el envío y recepción de tipos simples (cadenas, enteros, etc.). Si se requiere enviar objetos JavaBean o de tipo *custom* (**Vector**, **List**, etc.) se deben generar los deserializadores o serializadores correspondientes, es decir, las clases que permiten pasar un objeto XML a Java y viceversa.

La invocación de la aplicación cliente al *workflow* es síncrona y se hace a través del modelo estático de JAX-RPC. La Figura 5.9 muestra la forma en que se efectúa dicha invocación.

5.8.2. Pasos para la invocación del *workflow*

Antes de realizar la invocación del *workflow* se debe contar con el código cliente. Este es generado con la herramienta *WSDL2WebService* del Web Service Developer Pack. A través de esta utilería se generan los *stubs* y serializadores y deserializadores correspondientes.

Una vez que se cuenta con ellos se procede a crear los objetos que representan al servicio remoto. Entonces, se manipulan tal y como si residieran en la computadora cliente. Esto es, se pasan los parámetros a los métodos disponibles y se recupera la información poniéndola en un objeto del tipo que se solicita. A continuación se obtiene la información que se encuentra en el objeto que se envía en respuesta.

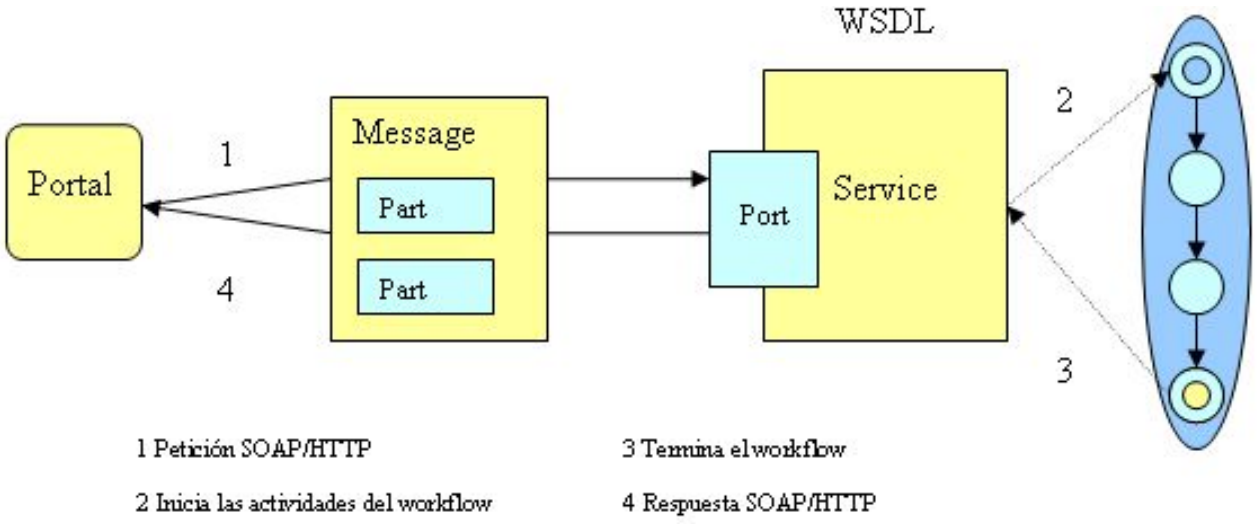

Figura 5.9: Invocación síncrona del workflow.

5.9. Conclusiones

En este capítulo se han visto todas las tecnologías usadas para implementar el APN. Java es el lenguaje que se ha usado para la implementación del APN, mientras que las conexiones a la base de datos de Access se han hecho con el puente JDBC/ODBC. La inspección de los WSDL de los servicios Web se ha hecho con una herramienta muy poderosa llamada WSIF. La manipulación de las plantillas se ha hecho con librerías de Apache que mapean documentos XML a objetos Java. La manipulación de la información se hace con implementaciones de XPath provistas por Oracle y el W3C. Finalmemente, la comunicación con el *workflow* se logra a través de JAX-RPC de Apache.

El uso de estas tecnologías es muy efectivo por varias razones. En primer lugar, son gratuitas (excepto la máquina BPE4WS de Oracle), lo que implica un ahorro económico enorme para la realización del proyecto. Por otro lado, están recibiendo apoyo de importantes empresas orientadas al desarrollo de *software*, lo que garantiza su uso y aceptación generalizada entre los desarrolladores. Además de que ofrecen un alto rendimiento en comparación con otras herramientas tecnológicas.

Capítulo 6

Conclusiones

La presente tesis ha mostrado las ventajas del uso de los servicios Web en el contexto de los ambientes comerciales modernos. De hecho, existen enormes expectativas acerca del impacto que tendrán los servicios Web en los mercados digitales y en la sociedad actual. Una de las más importantes es que mejorará y simplificará el proceso de integración de la aplicaciones de las empresas. Los servicios Web permitirán a las organizaciones integrar aplicaciones a un bajo costo, consumo de tiempo y sin soluciones intensivas de mantenimiento. Esto provocará que cada vez se vean más y mejores formas de servicios en Internet. De ahí la importancia de la investigación concerniente a esta nueva forma de hacer cómputo distribuido.

En particular, el trabajo realizado en esta tesis está relacionado con la orquestación de los servicios Web. Se ha podido ver a lo largo de este trabajo que el concepto de orquestación es uno de los más importantes debido al ambiente inherentemente heterogéneo en el que se desarrollan las aplicaciones. En este sentido, la orquestación permite crear aplicaciones orientadas a servicios que facilitan la invocación de servicios Web externos y convierten

a dichas aplicaciones en proveedores de servicios.

Sin embargo, a pesar de las enormes cualidades de BPEL4WS, tiene limitaciones que hacen inexorable su uso para cierto tipo de aplicaciones. En particular, se ha tratado el problema de agregar dinámicamente los participantes a un proceso de negocio. La importancia de un problema como éste es notable en el contexto de ambientes que son altamente variables y variados. Se ha visto que diversos han sido los intentos por resolver esta problemática. Los intentos han sugerido el uso de la Web semántica a través de lenguajes como DAML-S. No obstante, en este trabajo se ha planteado una solución más simple aunque sumamente útil: se ha propuesto un Administrador de Procesos de Negocio encargado de la ubicación, concretización y ejecución de plantillas BPEL4WS que describen formas recurrentes de procesos de negocio.

6.1. Contribuciones

Este trabajo tiene contribuciones en lo que se refiere a la automatización de procesos comerciales. Los resultados a los que se han llegado son:

- El diseño de procesos de negocio genéricos llamados plantillas. Estas plantillas se escribieron en BPEL4WS y abstraen procesos recurrentes al interior de una organización.
- El diseño e implementación de un Administrador de Procesos de Negocio. Su función es la de ubicar, recuperar, concretizar y ejecutar las plantillas en la máquina virtual de BPEL4WS de Oracle.
- Establecer los mecanismos para comunicarse con los *workflows* que se encuentran ejecutando en la máquina BPEL4WS.

- La construcción de un portal que se constituye como el enlace entre el usuario y el APN. Este portal tiene como objetivo recabar la información proveniente del usuario y pasársela al APN. Se ha construido con base en ontologías como RosettaNet y UN-SPSC que son ampliamente aceptadas en la industria.

- El diseño de un repositorio de plantillas BPEL4WS que funge como un almacén de procesos de negocio genéricos al cual puede acceder una empresa para incorporarlos a sus flujos de trabajo.

El APN ha sido probado con diversos casos de estudio que han permitido observar su utilidad. Concretamente se han mostrado los casos de estudio concernientes a la búsqueda de tiendas que cumplen con un criterio específico (problema conocido como *semantic matching*) como ofrecer el menor precio de un producto dado su código, ontología y cantidad, así como ejemplos derivados de éste como la búsqueda de empresas que ofrezcan la entrega de un producto al menor tiempo posible. Una vez que se han encontrado dichas empresas, un conjunto de peticiones se envía en paralelo a cada una de las tiendas también a través de la construcción dinámica de un *workflow* en BPEL4WS donde se integran cada una de ellas.

Como se mencionó, existen diversas propuestas que específicamente tratan el problema del *semantic matching*. La mayoría utiliza soluciones que implican el uso de herramientas semánticas como DAML-S, que, aunque son muy prometedoras, aún no han sido ampliamente probadas y se caracterizan por ser bastante complejas para los desarrolladores. En contraste, nuestra propuesta es bastante más simple y se corroboró que es adecuada para pequeñas empresas que buscan automatizar sus procesos. A través de los casos de estudio presentados en este trabajo se puede ver que si una empresa requiere un proceso de compra de productos en Internet que cumplan con un criterio específico, lo que tiene que hacer es seguir un proceso similar al planteado en esta tesis utilizando el APN desarrollado.

Nuestra propuesta resulta pues de gran utilidad para las pequeñas empresas que necesitan poner sus servicios en Internet en forma de servicios Web y requieren de un mayor dinamismo en la forma en que pueden agregar sus socios a sus procesos comerciales. Una herramienta como la que proponemos permitiría a las empresas simplemente definir sus procesos genéricos, y mediante una interfaz de usuario similar a la que también proponemos, agregarlos a la lógica del negocio.

Finalmente, la solución propuesta tiene implicaciones positivas en el desarrollo eficiente de procesos de negocio que son recurrentes al interior de una organización. La propuesta de crear un repositorio de plantillas BPEL4WS tiene como consecuencia inmediata la reutilización de código el cual, con ligeras modificaciones, puede adaptarse a necesidades diferentes pero que resuelven la misma clase de problemas (búsqueda o compra de productos, por ejemplo). Esto permite a las organizaciones reducir sus costos de desarrollo de software e incluso aumentar el valor de su empresa al comercializar no sólo productos tangibles sino también procesos de negocio a otras empresas que requieren de un proceso de negocio similar.

6.2. Trabajo a futuro

A pesar de los resultados alentadores que se han tenido en esta tesis, existen varios aspectos que no han sido cubiertos. Primeramente, el sistema ha sido probado para procesos de negocio simples. Sin embargo, para problemas de mayor complejidad el uso de tecnología como la relacionada con la Web semántica podría ser más adecuada.

En este mismo sentido, resulta importante analizar y tomar procesos de negocio reales que permitan establecer los alcances de nuestra propuesta. Ejemplos de estos podrían ser los del control de inventarios de una tienda comercial dedicada a la venta de productos por

Internet, los correspondientes a procesos industriales, entre otros. En el primer caso, es necesario equilibrar los niveles de inventario a partir de ciertos criterios cruciales para el proceso como el precio o el tiempo de entrega. En función de los resultados que arrojen las consultas a los servicios Web provistos por los diferentes proveedores, se decidirá cuales son los que se agreguen al proceso de negocio. El segundo caso podría presentar restricciones más complejas. Se podría pensar en una empresa que manufactura componentes electrónicos. Las restricciones podrían no ser simplemente con respecto al precio y tiempo de entrega sino también a la compatibilidad de los productos. En cualquier caso, el problema de agregar dinámicamente los participantes a un proceso de negocio es vital. Por lo que valdría la pena estudiar el sistema en procesos de este tipo.

Por otro lado, actualmente se está contemplando la posibilidad de dotar al APN de la capacidad de agregar no sólo servicios Web a un *workflow* como se hizo en este trabajo, sino también *workflows* o partes de un *workflow* a otro *workflow*. Esto con el fin de reutilizar procesos ya probados e incorporarlos a otros procesos. El lenguaje que permitirá la orquestación de servicios Web compuestos es el Web Services Choreography Description Language (WS-CDL). Este lenguaje permite representar intra e inter *workflows* empresariales gracias a que define el comportamiento entre las entidades de una empresa y entre sus federados. El proceso para integrar *workflows* a otros *workflows* sería similar al proceso descrito en este trabajo para agregar servicios Web a un *workflow*. Sin embargo, esto tendría mayores beneficios en la reutilización de *software* ya que no sólo se agregarían servicios modulares sino también actividades comerciales completas del negocio.

Finalmente, el APN es un tanto limitado en el sentido de que no provee herramientas que faciliten la inserción automática de plantillas. Hasta el momento, si se desea agregar una nueva plantilla para que se ejecute en la máquina BPEL4WS, es necesario hacerlo manualmente. La creación de un portal que administre el almacenamiento de las plantillas en el APN es importante porque ayudaría agregar, eliminar y actualizar fácilmente las

plantillas escritas en BPEL4WS.

Apéndice A

Artículos publicados

1. César Sandoval Hernández, Giner Alor Hernández, José Oscar Olmedo Aguirre. Dynamic generation of organizational BPEL4WS workflows. International Conference on Electrical and Electronics Engineering and X Conference on Electrical Engineering (ICEEE-CIE 2004). Proceedings ICEEE-CIE 2004, IEEE Press, ISBN: 0-7803-8532-2.

2. César Sandoval Hernández, Giner Alor Hernández, José Oscar Olmedo Aguirre. Generación Dinámica de GUIs para la invocación de servicios Web publicados en nodos UDDI, Recientes Avances en la Ciencia de la Computación en México, Research on Computing Science, Eds. Gilbukh A., Sidirov G., Olán W., Vera J., Vol. 7, ISBN 970-36-0149-9, pp. 68-79, CIC-IPN, México 2004.

3. Giner Alor Hernández, César Sandoval Hernández, José Oscar Olmedo Aguirre. Descubrimiento Dinámico de Servicios Web en nodos UDDI mediante USML, Recientes Avances en la Ciencia de la Computación en México, Research on Computing Science Series, Eds. Gilbukh A., Sidirov G., Olán W., Vera J., Vol. 7, ISBN 970-36-0149-9, pp. 56-67, CIC-IPN, México 2004.

4. Giner Alor Hernández, César Sandoval Hernández, et al., BPIMS- Business Process Integration and Monitoring for B2B E - commerce, XI International Congress on Computer Science Research, CIICC 2004, Proceedings CIICC 2004, ISBN: 968-5823-10-3. pp. 77-84.

5. Giner Alor Hernández, César Sandoval Hernández, José Oscar Olmedo Aguirre. BPIMS-WS: Brokering Architecture for Business Processes Integration for B2B E-commerce. IEEE International Conference on Electronics, Communications, and Computers. (IEEE CONIELECOMP 2005). Proceedings IEEE-CONIELECOMP 2005. IEEE Press. Aceptado para publicación

Apéndice B

Acrónimos y términos usados

1. BPEL4WS: El Business Process Execution Language for Web Services es un lenguaje composicional basado en XML que permite definir los procesos de negocio en términos de actividades de coordinaci'on y comunicación.

2. DAML-S: El DARPA Agent Markup Language es un lenguaje ontológico basado en DAML+OIL orientado a los servicios Web. Describe semánticamente las propiedades y capacidades de los servicios Web de tal forma que sean interpretables por las computadoras.

3. RDF: El Resource Description Framework es un lenguaje basado en XML que se usa para anotar el contenido de una página Web. Además, permite codificar, intercambiar y reutilizar información estructurada y publicar vocabularios comprensibles para las personas y procesables por las máquinas.

4. SOAP: El Simple Object Access Protocol es un protocolo ligero situado encima de protocolos de transporte como HTTP o SMTP. Se usa para enviar mensajes y hacer llamadas a procedimientos remotos. Su estructura está constituida por un sobre, una

cabeza y un cuerpo. En este último se introduce la información enviada o recibida de un servicio Web.

5. UDDI: El Universal Descrption, Discovery and Integration define un registro y protocolos asociados para la búsqueda y localización de servicios Web. Provee dos tipos de servicio: publicación y consulta. El primero define operaciones para registrar, modificar y eliminar negocios y servicios. Mientras que el segundo define operaciones que permiten realizar búsquedas sobre negocios y servicios.

6. WSIF: El Web Services Invocation Framwork es una API escrita en Java para la invocación de servicios Web no importando dónde y cómo se proveen los servicios. Contiene utilerías que permiten la inspección de los documentos WSDL de un servicio Web.

7. WSDL: El Web services Description Language es un lenguaje basado en XML, desarrollado por Microsoft e IBM, para describir servicios Web como colecciones de puntos de comunicación donde los mensajes entre el proveedor de servicios y el cliente son intercambiados. En esencia, el WSDL describe las interface de un servicio Web y provee la información de constacto para sus usuarios.

8. XML: El Extensible Markup Language es un conjunto de reglas orientadas a la creación de lenguajes de marcado semánticamente ricos. Se ha consolidado como el lenguaje estándar para la descripción de información promoviendo así una mayor interoperabilidad entre las aplicaciones.

9. XPath: Es un lenguaje usado para denotar y extraer fragmentos de documentos XML. Provee un medio efectivo para acceder a la información intercambiada entre los servicios Web.

10. Coreografía: La coreografía consiste en permitir a cada una de las partes involucradas en el proceso de negocio describir el rol que juegan en la interacción. En la coreografía

se deja a cada una de las partes del proceso ejecutarse desde su perspectiva sin que ninguna de ellas controle la conversación.

11. Intermediario: Es una aplicación que ofrece diversos servicios Web con los que se comunica el APN para obtener información acerca de otros servicios Web proveedores de diversos servicios como compra y consulta de productos. Estos proveedores se encuentran registados en un repositorio UDDI privado.

12. Ontología: Es un conjunto de definiciones inteligibles para las máquinas que permiten crear una taxonomía de clases y subclases de objetos y relaciones entre ellas. Cada clase y subclase tiene un significado semántico claro la aplicación cliente y el servicio Web.

13. Orquestación: La orquestación de servicios Web consiste en proveer los medios necesarios para establecer un proceso de negocio ejecutable que permita la interacción entre servicios Web internos o externos de una organización. El proceso de negocio es dirigido por una de las partes involucradas en dicho proceso.

14. Proceso de negocio. Un proceso de negocio consiste en la especificación del orden de ejecución de operaciones de una colección de servicios Web, la información compartida entre estos servicios Web, los socios involucrados y la forma en que éstos se involucran en el proceso de negocio.

15. RosettaNet: Es un consorcio que tiene como objetivo crear e implementar estándares que sirvan para formar un lenguaje de negocio común en la que los socios de un proceso de negocio puedan incorporarse libremente a una cadena de suministro.

16. Socio de negocio: Es una aplicación en forma de servicio Web que representa a un socio comercial de una organización.

17. UN-SPSC: El United Nations Standard Products and Services Code es una una

ontología promovida por las naciones unidas que facilita la clasificación de productos y servicios a través de estándares globales y abiertos.

18. Workflow: Un *workflow* consiste en la automatización de un proceso de negocio, parcial o total, durante la cual los documentos, información o tareas se transfieren de un participante a otro de acuerdo a un conjunto de reglas de procedimiento.

Bibliografía

[1] Barners-Lee T. et al., The Semantic Web, Scientific American, May 2001.

[2] Yergeau F. et al., Extensible Markup Language (XML) Recommendation Version 1.1, W3C, World Wide Web Consortium, February 2004, http://www.w3.org/XML/

[3] Brickley D., Guha R. Resource Description Framework (RDF) Specification, W3C, 2004, http://www.w3.org/RDF/

[4] Distributed Component Object model (DCOM) Home Page, Microsoft, http://www.microsoft.com/com/tech/DCOM.asp

[5] Common Object Request Broker Architecture (CORBA) Home Page, Object Management Group, http://www.corba.org/

[6] Bellwood T. et al., Universal Description, Discovery and Integration (UDDI) 3.0, Published Specification, Organization for the Advancement of Structured Information Systems (OASIS), July 2002, http://www.uddi.org/specification.html.

[7] Box D. et al., Simple Object Access Protocol (SOAP) 1.1, W3C Note 08, World Wide Web Consortium, 2000, http://www.w3.org/TR/SOAP/

[8] World Wide Web Consortium, 2004, http://www.w3.org.

[9] Chinnini R. et al., Web Services Description Language (WSDL) Specification, W3C, 2004, http://www.w3.org/TR/wsdl20.

[10] Little M, Webber J., Introducing BPEL4WS 1.0. Building on WS-Transaction and WS-Coordination, Web Services Journal, vol. 3 Issue 8, August 2003, pp 28-33.

[11] Peltz C., Web Service Orchestration and Choreography, Web Services Journal, vol. 3 Issue 7, July 2003, pp. 30-35.

[12] Andrews T., Curbera F., et al., Business Process Execution Language for Web Services (BPEL4WS) Version 1.1, IBM Software Group; 2003.

[13] Fischer L., Workflow handbook, Ed. Future Strategies Inc., E.U.A, 2001.

[14] Curbera F., Duftler M., Khalaf R., Nagy W., Mukhi N., y Weerawarana S., Unraveling the Web Services Web An introduction to SOAP, WSDL, and UDDI, IEEE Internet Computing, 2002.

[15] Daconta M., Obrst L., Smith K., The semantic Web: A guide to the future of XML, Web Services, and Knowledge Management, Ed. Wiley Publishing Inc., E.U.A, 2003

[16] James Clark, Steve DeRose, XML Path Language (XPath) Version 1.0, W3C, 1999.

[17] Gudgin M. et al., Simple Object Access Protocol (SOAP) Recommendation Version 1.2, W3C, 2003, http://www.w3.org/TR/soap12-part1/

[18] Thompson H., et al., XML Schema Part1: Structures Recommendation, W3C, 2001, http://www.w3.org/TR/xmlschema-1/

[19] Thatte Satish, Web Services for Business Process Design (XLANG), Microsoft Corporation; 2001.

[20] Leymann F., Web Services Flow Language (WSFL) Version 1.0, IBM Software Group; 2001.

[21] Web Services Coordination, WS-C 1.0 Specification, IBM 2002. Design. 2001 Microsoft Corporation.

[22] Cabrera F. et al., Web Services Transaction (WS-Transaction) Specification, IBM, 2002, http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/.

[23] Little M., Webber J., Introducing BPEL4WS 1.0 Building on WS-Transaction and WS-Coordination, Web Services Journal, 2003, pp 28-33.

[24] DAML-S Coalition, DAML-S: Web Service Description for the Semantic Web, Proc. Intl Semantic Web Conf. (ISWC), LNCS 2342, Springer Verlag, 2002, pp. 348-363.

[25] Paolucci M., Sycara K., Automonous Semantic Web Services, IEEE Computing, 2003, pp.34-41.

[26] Verma K., et al, On accommodating Inter Service Dependencies In Web Process Flow Composition, AAAI Spring Symposium Series, 2004.

[27] Jung J., et al., Business Process Choreography for B2B Collaboration, IEEE Internet Computing, pp. 37-45, 2004.

[28] Laukkanen M., Helin H., Composing Workflows of Semantic Web Services, AAMAS Workshop on Web services and agent-based engineering, 2003.

[29] Alor G., Olmedo O., Sistema de Intermediación para el Comercio Electrónico B2B Basado en Servicios Web, XII Congreso Internacional en Computación, Avances en Ciencias de la Computación, Research on Computing Science Vol. 3, ISBN 970-36-0098-0, pp 330.

[30] RosettaNet Lingua Franca for eBusiness Home Page, 2004, http://www.rosettanet.org/

[31] Amstrong E. et al, The Java Web Services Tutorial, Sun Microsystems, 2003.

[32] Jablonski Stefan, Bussler Christoph, Workflow Management, Modeling concepts, architecture and implementation, Thomson Computer Press, 1996.

[33] Web Service Invocation Framework (WSIF), Apache Software Foundation, 2004, http://ws.apache.org/wsif/.

[34] Web Services Description Language for Java (WSDL4J), IBM, 2004, http://www-124.ibm.com/developerworks/projects/wsdl4j/.

[35] Universal Discovery, Description and Integration for Java (UDDI4J), IBM, 2004, http://www-124.ibm.com/developerworks/oss/uddi4j/.

Los abajo firmantes, integrantes del jurado para el examen de grado que sustentará el **Sr. César Sandoval Hernández**, declaramos que hemos revisado la tesis titulada:

GENERACIÓN DINÁMICA DE WORKFLOWS ORGANIZACIONALES ESCRITOS EN BPEL4WS

Y consideramos que cumple con los requisitos para obtener el Grado de Maestría en Ciencias en la especialidad de Ingeniería Eléctrica opción Computación.

Atentamente,

Dr. José Oscar Olmedo Aguirre

Dr. Pedro Mejía Álvarez

Dr. Arturo Díaz Pérez