Tesina de máster:

Herramienta para la generación y despliegue de composiciones de servicios Web mediante modelos BPMN

Javier De La Fuente Sales

Julio de 2012, valencia

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Máster en Ingeniería del Software, Métodos Formales y Sistemas de la Información

Director: Vicente Pelechano Ferragud
Subdirector: Germán Harvey Alférez Salinas

Tabla de contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1 Introducción

Actualmente, se está viendo un gran crecimiento en el uso, tanto de la Service-Oriented Architecture (SOA) [**1**], como de servicios Web. De hecho, el 62% de las empresas están utilizando o quieren utilizar SOA en un futuro próximo [**2**].

SOA representa un modelo de arquitectura que tiene como objetivo mejorar la agilidad y la rentabilidad de una empresa. Esta propuesta es atractiva para las empresas, pues éstas pueden conectar nuevos servicios o mejorar los existentes haciendo frente a los nuevos requerimientos del negocio en un ambiente cambiante.

Los servicios Web son la implementación más frecuente de SOA [**3**]. Éstos proporcionan una base técnica adecuada para la comunicación de los procesos de negocio, dentro y fuera de la empresa. Sin embargo, para apoyar adecuadamente los procesos de negocio y su gestión, es necesario un mecanismo para describir como estos servicios Web van a ser utilizados [**4**].

Es en este contexto surgió el Business Process Execution Language (WS-BPEL) [**5**], el cual es uno de los lenguajes más utilizados para la definición de procesos basados en servicios Web a nivel empresarial actualmente [**6**]. Una de las pruebas de ello, es la gran cantidad de herramientas desarrolladas para dar soporte en el desarrollo y ejecución de procesos WS-BPEL [**7**].

No obstante, el principal problema de WS-BPEL para la definición de procesos de negocio, es que éste es un lenguaje complejo, que requiere de personal con un amplio conocimiento de programación. Esto es totalmente opuesto a la gestión tradicional empresarial de los procesos de negocio, la cual es llevada a cabo por administradores de negocio, que carecen generalmente de conocimientos de programación. Además, los procesos de negocio deberían ser fácilmente entendibles por los clientes involucrados en éstos.

Asimismo, para mantener la competitividad de la empresa con respecto a sus competidores, sus procesos de negocio deben estar totalmente operativos en todo momento y en constante mejora. Esto implica una gran cantidad de cambios en el proceso WS-BPEL. Debido a sus características técnicas, hacer estos continuos cambios en WS-BPEL requiere tiempo, lo cual en muchas ocasiones, marca la diferencia entre el éxito y el fracaso. Además la realización de estos cambios y la inclusión de nuevos procesos no deben afectar a los procesos que ya están en funcionamiento y que no se pueden parar.

Por los motivos mencionados anteriormente, se propone aplicar a la definición de proceso en WS-BPEL, el desarrollo dirigido por modelos (MDD) [**8**]. MDD es un paradigma de desarrollo software, que usa modelos como principal elemento del desarrollo. Este hecho facilita el desarrollo, reusabilidad y mantenibilidad del software enormemente.

Concretamente, para la implementación de la propuesta se ha utilizado un modelo principal basado en el Business Process Execution Language (BPMN) [**9**] como modelo principal. BPMN proporciona una notación gráfica para la definición de procesos de negocio, la cual es fácilmente entendible sin necesidad de conocimientos de programación. Con ello se consigue elevar el nivel de abstracción y mejorar la rapidez con la que se pueden enfrentar los cambios en los procesos de negocio.

Las principales aportaciones de la propuesta, dentro del marco explicado, son:

1) Generación de procesos WS-BPEL totalmente operativo mediante una aproximación basada en modelos. Con ello se consigue tener procesos de negocio implementados sobre el lenguaje WS-BPEL.

2) Despliegue del proceso WS-BPEL generado en caliente sin necesidad de reiniciar el servidor. Con esto se consigue que la modificación de un proceso no afecte al resto de procesos en ejecución y por tanto se evitan interrupciones innecesarias de los servicios de la empresa.

3) Herramienta para la definición de procesos WS-BPEL basada en modelos BPMN. Esta herramienta permite al usuario la introducción de la información especifica de los servicios de una formal fácil e intuitiva.

Además de las principales aportaciones comentadas, la tesina ha sido desarrollada como futura extensión para le herramienta Moskitt [**10**]. Esta herramienta se centra principalmente en el desarrollo de software dirigido por modelos. Moskitt está apoyada por la Consellería de Infraestructuras, Territorio y Medio Ambiente (actualmente se esta preparando su uso en el resto de Consellerías del Gobierno Valenciano).

A continuación se presentan: 1) la motivación que ha llevado a la realización de esta propuesta; 2) los principales objetivos que se han querido conseguir con la elaboración de la propuesta; 3) la problemática que se ha querido solventar con la implementación de la propuesta; 4) la solución implementada; y 5) se explica la estructura que sigue la memoria.

1.1. Motivación

Muchos motivos han animado el desarrollo de esta propuesta: la gran cantidad de motores y herramientas para el desarrollo y ejecución de procesos basados en WS-BPEL (Apache ODE [**11**], Oracle BPEL [**12**], Microsoft Biztalk [**13**], etc.), las ventajas de WS-BPEL sobre lenguajes clásicos como Java o C#, la abundancia de investigaciones y propuestas entorno a la generación de procesos WS-BPEL a partir de modelos BPMN [**14**,**15**] y la existencia de BPMN y las ventajas que aporta el uso de una notación gráfica (tales como, abstracción, sencillez, facilidad de entendimiento). No obstante, los motivos principales han sido cuatro:

En primer lugar, la creación de una herramienta orientada al usuario final, que permita construir procesos de negocio basados en WS-BPEL a partir de modelos BPMN. Con ello se consigue hacer la construcción de procesos WS-BPEL más sencilla y accesible a un mayor número de personas. Además, mediante la utilización de modelos se consiguen procesos más fiables con menor posibilidad de errores y más fácilmente modificables y adaptables. Esta última característica es esencial para conseguir procesos de negocio exitosos.

En segundo lugar, la existencia de BABEL [**15**], un proyecto que genera código WS-BPEL incompleto a partir de modelos BPMN, cuyos procesos pueden ser fácilmente completados mediante la introducción de información por el usuario.

En tercer lugar, la facilidad con la que motores de ejecución de procesos de negocio WS-BPEL como Apache ODE [**11**] permiten el despliegue de los procesos en caliente, sin necesidad de reiniciar el servidor sobre el que se ejecutan.

Por último, generar una herramienta capaz de ser extendía e incorporada fácilmente sobre la herramienta Moskitt y que permita agilizar y mejorar el uso que el Gobierno de Valencia y la propia Consellería hacen de los motores de procesos WS-BPEL. Además el Departamento de Organización hace uso de BPMN para especificar los procedimientos administrativos que se deben implementarse en la Consellería. Por lo que la incorporación de la herramienta aquí presentada no supondría un gran cambio en la manera de trabajar de estas administraciones, pero si supondría un aumento de la agilidad con la que se trabajan con este tipo de procesos.

Todo ello, ha llevado a la implementación de una herramienta para la generación y despliegue de composiciones de servicios Web mediante modelos BPMN.

1.2. Objetivos

El objetivo principal de esta propuesta, es implementar un framework para el desarrollo de procesos WS-BPEL totalmente operativos, desplegados en caliente y de cara al usuario final usando MDD. Este objetivo se puede descomponer en una serie de sub-objetivos:

1) Implementar la transformación de modelos BPMN a procesos WS-BPEL.
2) Desarrollar una herramienta que permita completar el WS-BPEL generado en BABEL usando MDD.
3) Desplegar el proceso WS-BPEL generado en caliente.
4) Creación de una herramienta fácilmente incorporable a la herramienta Moskitt.

5) Crear un caso de estudio complejo y realista que permita demostrar la utilidad, facilidad de la aproximación desarrollada.

1.3. Problemática

Cada vez las empresas optan más por arquitecturas SOAs basadas en servicios Web. Sin embargo, la implementación más habitual de estos procesos está basada en WS-BPEL, un lenguaje técnicamente complejo, que requiere amplios conocimientos de programación.

Lo anterior crea dos grandes problemas:

1) El personal que define generalmente los procesos de negocio en una empresa no tiene conocimientos suficientes para trabajar con este lenguaje.

2) Los procesos de negocio son cambiantes. Realizar cambios implica tiempo, reiniciado de servidores, etc. Una empresa no se puede permitir dejar todos sus servicios parados en ningún momento.

1.4. Solución propuesta

Con el objetivo de solventar los problemas mencionados en la apartado 1.3, se ha desarrollado una herramienta para la generación de procesos de negocio WS-BPEL a partir de modelos BPMN. Estos modelos ya han sido comúnmente utilizados en las empresas, y proporcionan una notación gráfica fácil de entender por las personas involucradas en el proceso.

Con el fin de realizar la transformación de BPMN a WS-BPEL se ha utilizado el proyecto BABEL [**15**] el cual proporciona una solución sencilla para la transformación de modelos BPMN en código WS-BPEL. No obstante, BABEL no genera código WS-BPEL completo. Por lo tanto es necesario introducir información referente a los servicios Web, al despliegue final, etc. Para facilitar la inclusión de esta información, se ha desarrollado una herramienta que permite introducir los datos que no genera BABEL de una manera sencilla y atractiva para el usuario.

Utilizando un modelo BPMN como base inicial e incorporando la información específica de los servicios Web a posteriori se consiguen procesos definidos independientemente de los servicios Web finales que lo implementarán. Esto permite tener una mayor facilidad para gestionar los cambios, especialmente los provocados por fallos de los servidores. Por ejemplo, si un servicio Web falla, sólo es necesario cambiar la información referente a este servicio por otro servicio Web con similar funcionalidad y generar el nuevo proceso con la nueva información.

Además de generar procesos WS-BPEL completos, la aproximación presentada genera automáticamente todos los documentos que se necesitan para la ejecución y

despliegue del proceso. Asimismo, el despliegue es llevado a cabo automáticamente en el servidor en caliente. Es decir, no es necesario reiniciar el servidor para realizar los cambios necesarios.

1.5. Estructura de la memoria

En este apartado se explica resumidamente el contenido de la tesina.

Capítulo 2: Visión general de la propuesta

Este capítulo presenta los conceptos básicos, técnicas y la metodología claves para el entendimiento de la propuesta, explicando sus características más importantes de forma que el desarrollo de la propuesta sea más sencillo de comprender. Además, se explica el desarrollo que se ha seguido para la elaboración de la tesina y por último se presentan las tecnologías que se han utilizado para desarrollar la tesina.

Capítulo 3: Generación dirigida por modelos y despliegue en caliente de procesos BPEL

En este capítulo, se presenta el desarrollo completo y detallado de la propuesta. Se comienza con una visión global de todas las transformaciones implementadas y de sus interacciones. Después se explica cada una de las transformaciones en detalle. Por último se habla del despliegue en caliente.

Capítulo 4: Caso de estudio (agencia de viajes)

Se presenta un caso de estudio de una agencia de viajes para ejemplificar el uso de la propuesta de forma que se muestre en unas situaciones concretas la funcionalidad del trabajo. Además, se evalúa el desempeño y la usabilidad de la propuesta implementada para este caso de estudio.

Capítulo 5: Trabajos relacionados

En este capítulo se describen trabajos, herramientas e investigaciones relacionadas con la propuesta aquí explicada. Se comienza explicando transformaciones similares a BABEL, que generan código WS-BPEL incompleto. Después se explican tres de las propuestas más relevantes, que proponen una solución para la generación de procesos WS-BPEL completos.

Capítulo 6: Conclusión y trabajos futuros

En este capítulo se presentan las contribuciones principales del desarrollo que se ha realizado, los resultados y conclusiones obtenidas. Asimismo, se presentan las posibles líneas de trabajos futuros y los diferentes aspectos relacionados con la tesina que no se han abordado y que se podrían llevar a cabo en desarrollos futuros.

Capítulo 2 Visión general de la propuesta

En este apartado se muestra una visión panorámica de la propuesta. Para ello se describen elementos que la componen, presentando los conceptos principales necesarios para entender el desarrollo de la propuesta que se presenta en el Capítulo 3.

Primero, se presentan los conceptos básicos necesarios para entender el proceso de desarrollo en el que se enmarca la tesina. Seguidamente, se presenta el desarrollo de la propuesta. Por último, se presentan todas las tecnologías utilizadas para la implementación de la propuesta.

2.1. Conceptos básicos

En este apartado se describen de forma detallada los conceptos básicos y la terminología necesaria para el entendimiento de la propuesta. En primer lugar, se introduce el desarrollo dirigido por modelos, elemento principal y guía para el desarrollo de toda la propuesta; en segundo lugar, se explica la arquitectura SOA y los diferentes roles y actividades que la componen; en tercer lugar, se definen los servicio Web, los estándares que lo componen y cómo funcionan dentro de una arquitectura SOA; en cuarto lugar, se habla de BPMN y de los diferentes elementos que forman su notación; y por último, se explica WS-BPEL, su función, estructura, y las actividades que lo forman.

2.1.1. Desarrollo dirigido por modelos

El Desarrollo de Software Dirigido por Modelos (MDD son sus siglas en inglés) constituye una aproximación para el desarrollo de sistemas software donde se persigue elevar el nivel de abstracción en el desarrollo de software dándole una mayor importancia al modelado conceptual y al papel de los modelos en el desarrollo de software actual [**16**].

Una de las propuestas más utilizadas de MDD es la Model Driven Architecture (MDA). Ésta es una aproximación para el desarrollo de sistemas software, basada en la separación entre la especificación de la funcionalidad esencial del sistema y la implementación de dicha funcionalidad usando plataformas de implementación específicas [**17**]. En MDA, los modelos se pueden clasificar en tres niveles:

1) Platform Independent Model (PIM): Es una descripción de la funcionalidad del sistema en forma independiente de las características de plataformas de implementación específicas.

2) Platform Specific Model (PSM): Es una descripción del sistema en términos de una plataforma específica.

3) Implementation Specific Model (ISM): Es una descripción (especificación) del sistema a nivel de código.

Básicamente, la idea es transformar de forma automática o semiautomática los modelos de mayor abstracción en modelos cada vez más dependientes de la implementación hasta llegar al código final. En la Figura 1 se puede ver la estructura clásica que proporciona MDA.

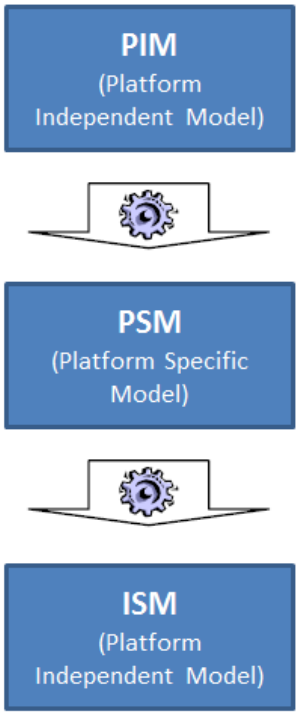

Figura 1: Model Driven Architecture

Algunos de los beneficios más importantes que proporciona la utilización de MDA son [**17**]:

- Reducción de costes desde el principio hasta el fin.
- Reutilización de código, aplicaciones y experiencia.
- Representación independiente de la tecnología.
- Escalabilidad y robustez.
- Interoperabilidad entre diferentes tecnologías.

2.1.2. Arquitectura orientada a servicios (SOA)

Un servicio es un recurso abstracto, que representa una capacidad de realizar tareas que conforman una funcionalidad coherente, desde el punto de vista de las entidades proveedoras y consumidoras [**18**]. Basado en este principio, Service-Oriented Architecture (SOA) [**1**] define una arquitectura de software, donde todas las tareas y procesos de software son implementados como servicios para ser consumidos sobre una red.

Propiedades

La W3C define SOA, como una arquitectura para sistemas distribuidos, caracterizada por las siguientes propiedades [**19**]:

- Vista Lógica: El servicio es una abstracción (vista lógica) de los programas, bases de datos, procesos de negocio, etc., definido en términos de lo que hace (llevando a cabo una operación de negocio).
- Orientación a Mensajes: El servicio se define formalmente en términos de los mensajes intercambiados entre agentes proveedores y solicitantes, y no está basado en las propiedades de los agentes. La estructura interna del agente (lenguaje de programación, base de datos, proceso, etc.) se abstrae en SOA. Esto permite incorporar cualquier componente o aplicación a esta arquitectura.
- Orientación a la Descripción: Un servicio es una abstracción que define lo que son capaces de hacer las bases de datos, procesos de negocio, etc.
- Granularidad: Los servicios deben utilizar un número reducido de operaciones con una complejidad relativamente grande en las operaciones llevadas a cabo.
- Orientación a la Red: Los servicios tienden a usarse a través de la red, aunque éste no es un requisito absoluto.
- Neutral a la Plataforma: Los mensajes se envían en un formato estándar y neutral a la plataforma.

Arquitectura SOA

La Figura 2 muestra los distintos roles que componen una arquitectura SOA y las tres acciones asociadas a ellos. Los roles, que se encuentran en una arquitectura SOA son:

- Proveedor de servicios: Es una entidad accesible a través de la red, que acepta y ejecuta consultas de consumidores, y publica sus servicios y su contrato de interfaces en el registro de servicios, para que el consumidor de servicios pueda descubrir y acceder al servicio. Un ejemplo de este rol es una entidad que hospeda un servicio Web accesible a través de una red.
- Solicitante de servicios: Cualquiera que desee consumir el servicio. Por ejemplo, una aplicación que consuma un servicio Web que retorna el estado del tiempo.
- Registro de servicios: Es un repositorio de las descripciones de los servicios ofrecidos por los proveedores. Éste tiene dos funciones: 1) permitir que los proveedores publiquen las descripciones de sus servicios; y 2) permitir que los consumidores

busquen estas descripciones. En otras palabras, un repositorio UDDI es un repositorio de servicios.

Las acciones, que pueden realizar cada uno de los roles, son:

- Enlazar: Crear la relación cliente-servidor, entre el solicitante del servicio y el proveedor del servicio. Puede ser dinámica, mediante el uso de un proxy o estática hecha a mano.
- Publicar: Publicación/registro del servicio en un repositorio.
- Buscar: Búsqueda por parte del consumidor de un servicio deseado dada una serie de criterios. El sistema de búsqueda es dependiente del repositorio.

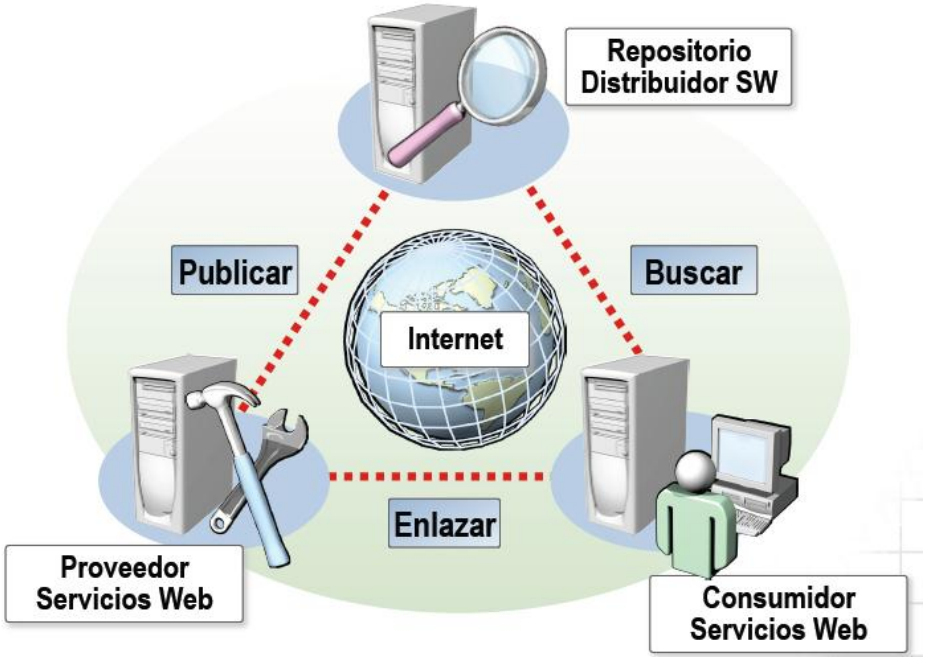

Figura 2: Arquitectura Orientada a Servicios (SOA) [20**]**

2.1.3. Servicios Web

Existen numerosas definiciones de lo que es un servicio Web [**21**,**22**,**23**], lo que muestra su complejidad a la hora de dar una definición que englobe todo lo que implica. Una de las más completas es la del W3C:

"Un servicio Web es un sistema software diseñado para soportar a la interoperabilidad máquina-máquina sobre la red. Este tiene una interfaz descrita en un formato procesable por una máquina (específicamente WSDL). Otros sistemas interactúan con el servicio Web de la manera especificada por su descripción utilizando mensajes SOAP,

por lo general transmitidos a través de HTTP con una serilización XML y unión con otros estándares relacionados con la Web" [**18**].

Conceptos base

Todas las definiciones que tratan de especificar qué es y de que está formado un servicio Web, tienen los siguientes conceptos básicos en común:

- Un servicio Web expone funcionalidad a un consumidor (una aplicación, persona o sistema que utilice el servicio Web) a través de la Web.
- Están basados en estándares de la Web, tales como HTTP, XML, SOAP, WSDL, UDDI, etc.
- Pueden implementarse bajo cualquier lenguaje y plataforma.
- Se pueden ver como cajas negras. Es decir, el sistema puede ser visto como un conjunto de entradas y salidas sin importancia de su funcionamiento interno.
- Permiten interconectar compañías, aplicaciones, dispositivos, etc.
- Distribuyen e integran la lógica de una aplicación.
- Están débilmente acoplados. Es decir, los servicios Web tienen una comunicación reducida entre ellos. Esta es una de las principales ventajas, que aportan los servicios Web, ya que permiten la reutilización de éstos en otros sistemas.

Protocolos y estándares

En la Figura 3 se muestra la pila de protocolos básica de los servicios Web.

Figura 3: Pila de protocolos básica de los servicios Web [24**]**

A continuación, se explica cada uno de los protocolos.

Universal Description, Discovery and Integration (UDDI)

UDDI [**25**] es un registro público diseñado para almacenar, de forma estructurada, información sobre empresas y los servicios que éstas ofrecen. A través de UDDI, se puede publicar y descubrir información de una empresa y sus servicios Web.

Lo más importante es que UDDI contiene información sobre las interfaces técnicas de los servicios de una empresa (WSDLs). A través de un conjunto de llamadas a APIs XML basadas en SOAP, se puede interactuar con UDDI, tanto en tiempo de diseño, como de ejecución, para descubrir datos técnicos de los servicios, que permitan invocar y utilizar los servicios Web. De este modo, UDDI sirve como infraestructura para una colección de software basado en servicios Web.

Web Services Description Language (WSDL)

WSDL es un lenguaje basado en XML para la descripción de servicios Web [**26**]. Éste permite describir la interfaz pública de los servicios Web, es decir, detalla los protocolos y los formatos de los mensajes necesarios para interactuar con el servicio Web que describe.

En cuanto a la estructura de un documento WSDL, la cual se muestra en la Figura 4, se pueden diferenciar dos partes: la parte abstracta del documento (independiente de la aplicación), y la parte concreta, que define los enlaces a protocolos y los puntos finales de acceso al servicio.

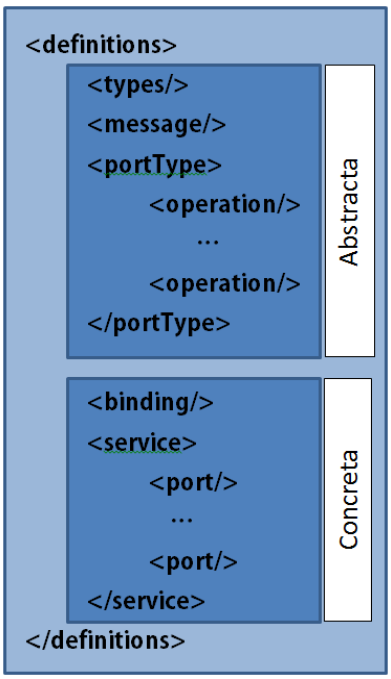

Figura 4: Estructura del documento WSDL

La parte abstracta consta de los siguientes elementos:

- Tipos de datos <*types*>: Proporciona las definiciones de tipos de datos utilizados para describir los mensajes intercambiados.

- Mensaje <*message*>: Es la definición abstracta de los datos que se transmiten. Un mensaje se compone de partes lógicas, cada uno de los cuales está asociado con una definición dentro de algún tipo de sistema.

- Operación <*operation*>: Descripción abstracta de una operación soportada por el servicio. Define un intercambio simple de mensajes. Al igual que en un lenguaje de programación común, una operación abstracta, consiste en la definición del nombre, las entradas, y la salida de la operación, sin especificar su lógica interna.

- Tipo de puerto <*portType*>: Conjunto de operaciones abstractas.

  La parte concreta consta de los siguientes elementos:

- Ligadura <*binding*>: Especifica los protocolos de comunicación usados.

- Servicio <*service*>: Define un conjunto de puertos relacionados.

- Puerto <*port*>: Define el punto de conexión a un servicio Web.

Simple Object Access Protocol (SOAP)

SOAP es un protocolo estándar, que define cómo dos componentes en diferentes procesos pueden comunicarse por medio del intercambio de datos XML [**27**]. SOAP fue creado por Microsoft, IBM y otros, y está actualmente bajo el auspicio del W3C. Éste es el protocolo para el intercambio de información utilizado por los servicios Web.

eXtensible Markup Languageble Markup Language (XML)

XML es un estándar utilizado para normalizar el intercambio de datos entre participantes, proporcionando un medio excelente para codificar y formatear datos [**28**]. Los elementos y atributos de XML, definen tipos y estructuras de información para los datos que llevan, incluyendo la capacidad de modelar datos y estructuras específicas a un dominio del sistema.

La sintaxis de XML usada en las tecnologías de los servicios Web es utilizada para representar los datos (WSDL), definir cómo y con qué calidad se transmiten los datos (SOAP) y detallar cómo se publican y descubren los servicios (UDDI).

Hypertext Transfer Protocol (HTTP)

HTTP es el protocolo más común para el intercambio de información en la Web [**29**]. La principal característica de HTTP es la tipificación y la negociación de la

representación de datos, permitiendo la construcción de los sistemas, de forma independiente de los datos que se transfieren.

Servicios Web como implementación de SOA

Aunque los servicios Web no necesariamente significan SOA, y no todas las SOAs están basadas en servicios Web, la relación entre las dos tendencias es importante, y se potencian mutuamente: el interés por los servicios Web lleva hacia SOA, y las ventajas de la arquitectura SOA ayudan a que las iniciativas de servicios Web tengan éxito [**30**].

Algunas de las ventajas que aportan los servicios Web son: 1) al contrario que CORBA [**31**] y DCE [**32**], los estándares de servicios Web no tienen detractores entre los fabricantes; 2) la flexibilidad de los Servicios Web para soportar aplicaciones multicanal; 3) la capacidad de SOAP de pasar por los firewalls, aprovechando la ubicuidad del HTTP; y 4) el soporte de servicios Web en servidores de aplicaciones que albergan lógica empresarial.

La arquitectura SOA, implementada con los estándares de los servicios Web, se ve en la Figura 5. En ella se puede observar como los diferentes roles y actividades, de los cuales se ha hablado en el apartado 2.1.2, son implementados por los estándares de los servicios Web.

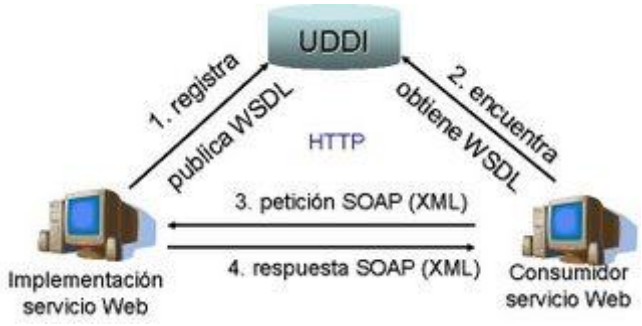

Figura 5: Servicios Web como implementación de SOA [33**]**

Utilizar los servicios Web como implementación para SOA significa utilizar SOAP como lenguaje de intercambio, WSDL como lenguaje para la descripción de los servicios y UDDI para la publicación o registro de los mismos.

La secuencia de ejecución es la siguiente: 1) el proveedor del servicio da de alta el servicio Web en el registro. Con este fin, el proveedor almacena en el registro el documento de descripción de éste (WSDL); 2) el solicitante del servicio busca en el registro un servicio Web que pueda adaptarse a sus necesidades; 3) una vez seleccionado el servicio, el solicitante lo invoca mediante el envío de un mensaje SOAP, en el cual se indica la acción a realizar y los datos de entrada; y 4) el servicio Web recibe la petición y ejecuta la funcionalidad. Para finalizar, el servicio Web envía un mensaje SOAP al solicitante con los resultados obtenidos [**33**].

2.1.4. Business Process Modeling Language (BPMN)

Business Process Model and Notation (BPMN) es un estándar desarrollado por el Object Management Group (OMG), cuyo principal objetivo es proveer una notación estándar que sea fácilmente legible y entendible por parte de todos los involucrados e interesados del negocio [**9**,**34**].

Elementos de un diagrama BPMN

Los elementos que forman un diagrama BPMN se pueden clasificar en cuatro categorías principales, que son: *Objetos de Flujo*, *Objetos de Conexión*, *Carriles de Piscina* y *Artefactos*. Para simplificar, sólo se hablará de los elementos relevantes para el completo entendimiento de la propuesta.

En la Figura 6, se puede ver un diagrama BPMN de ejemplo. Éste expresa la lógica asociada al pago interno realizado por una agencia de viajes a sus proveedores. Este diagrama es utilizado como base para la explicación de las categorías del diagrama BPMN usadas en este estudio.

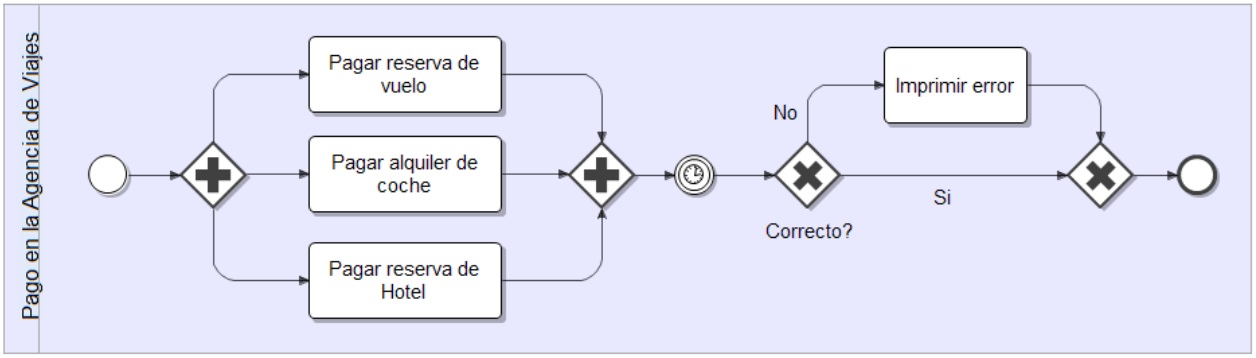

Figura 6: Diagrama BPMN que describe los pagos internos en una agencia de viajes

Carriles de nado

Los *Carriles de Nado* son un mecanismo visual de actividades organizadas y categorizadas. Éstos constan de dos elementos: *Piscina* y *Carril*.

La *Piscina* contiene al resto de elementos del diagrama, la cual se puede ver remarcada en la Figura 7. Ésta representa los participantes principales de un proceso, por lo general, separados por las diferentes organizaciones. Una *Piscina* puede contener uno o más carriles. No obstante, la transformación que realiza BABEL sólo soporta el uso de un *Carril*.

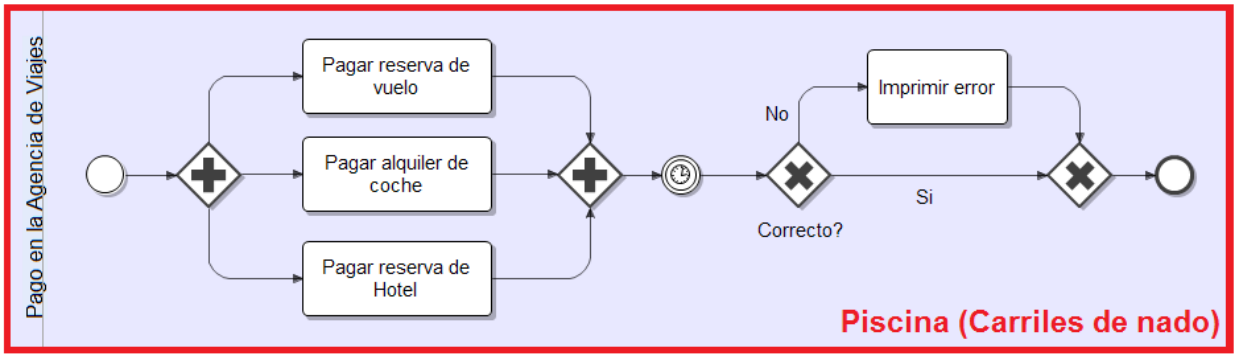

Figura 7: Diagrama BPMN que describe los pagos internos en una agencia de viajes con el remarcado del elemento piscina

Objetos de flujo

Los *Objetos de Flujo* son los elementos principales descritos dentro de BPMN y constan de tres elementos principales: *Eventos*, *Actividades* y *Compuertas*.

Los *Eventos*, son los encargados de describir lo que sucede dentro del proceso. Éstos se pueden clasificar en tres grupos según su posición dentro del proceso: *Iniciales*, *Intermedios* y *Finales*.

Los eventos soportados por la transformación BABEL son: 1) *Evento Inicial Vacío*: son utilizados para marcar el inicio del proceso; 2) *Evento Intermedio Temporizador*: permite iniciar la espera de un cierto tiempo antes de continuar con la ejecución del proceso; y 3) *Evento Final Vacío*: marca el final del proceso. En la Figura 8 se muestra el uso de estos eventos.

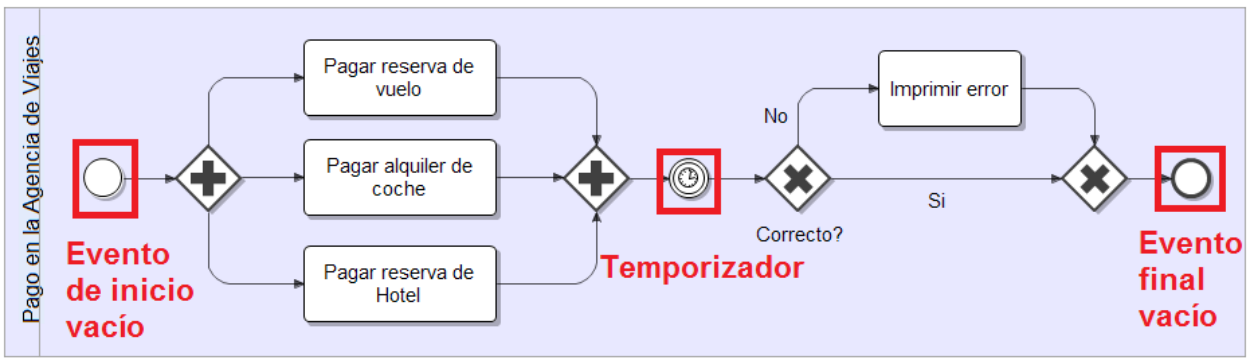

Figura 8: Diagrama BPMN que describe los pagos internos en una agencia de viajes con el remarcado de los elementos evento

Las *Actividades* se encargan de describir el tipo de trabajo realizado. En esta categoría existen dos elementos relevantes para la propuesta: 1) *Tarea simple*: su función es representar una sola unidad de trabajo que no se puede dividir en un mayor nivel de detalle. En la Figura 9 se muestran los elementos de tipo actividad utilizados para el pago interno; y 2) *Subproceso:* se utiliza para ocultar o mostrar otros niveles de detalle de los procesos de negocio. Cuando se minimiza un subproceso se indica con un signo más.

Cuando se expande el rectángulo redondeado se muestra todos los objetos de flujo, los objetos de conexión, y artefactos. Además, un *Subproceso* tiene sus propios eventos de inicio y fin. Asimismo, los flujos de proceso del proceso padre no deben cruzar la frontera. En la Figura 10 se puede ver una imagen que ilustra la utilidad y los dos posibles estados de este elemento.

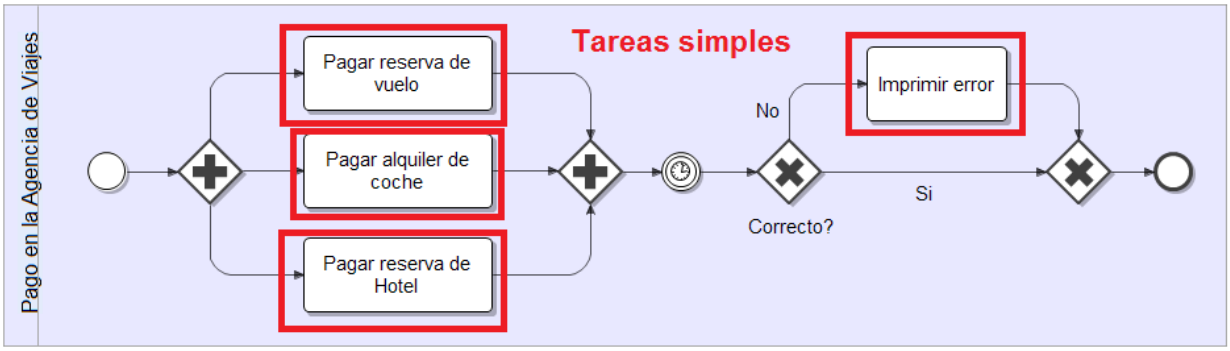

Figura 9: Diagrama BPMN que describe los pagos internos en una agencia de viajes con el remarcado de las actividades

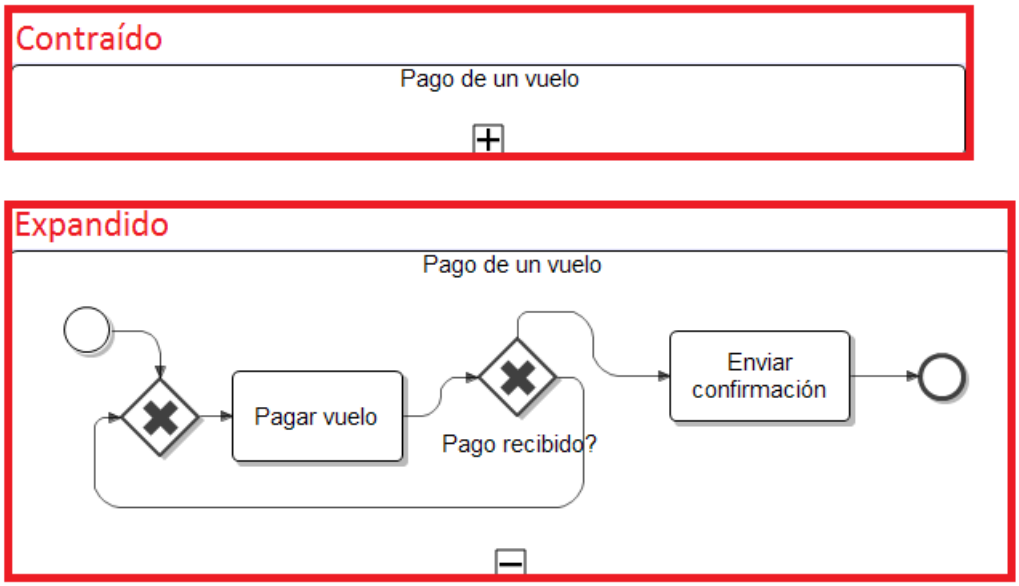

Figura 10: Diagrama BPMN que describe los dos estados posibles de un elemento de tipo subproceso

Las *Compuertas* están representadas por una figura de diamante y determinan, si se bifurcan o se combinan las rutas, dependiendo de las condiciones expresadas. Los elementos soportados de este tipo son: 1) *Compuertas Exclusivas*: permiten partir y unir caminos. Sólo uno de los diferentes caminos puede ser escogido (ver Figura 11); y 2) *Compuertas Paralelas*: puede utilizarse tanto para crear flujos paralelos, como para sincronizar diferentes flujos dentro de un proceso. Su significado depende de si se utiliza para partir un proceso en varios flujos o para juntarlos (ver Figura 12).

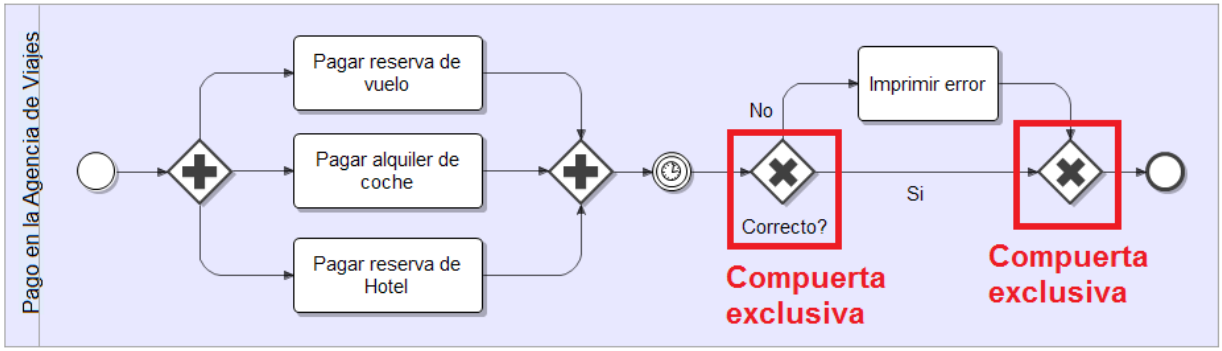

Figura 11: Diagrama BPMN que describe los pagos internos en una agencia de viajes con el remarcado de las compuertas exclusivas

Por ejemplo, en la Figura 12, la *Compuerta Paralela* más a la izquierda indica que se puede seguir cualquiera de los tres caminos. La que está más a la derecha indica que hasta que no se ejecuten los tres caminos no se puede avanzar.

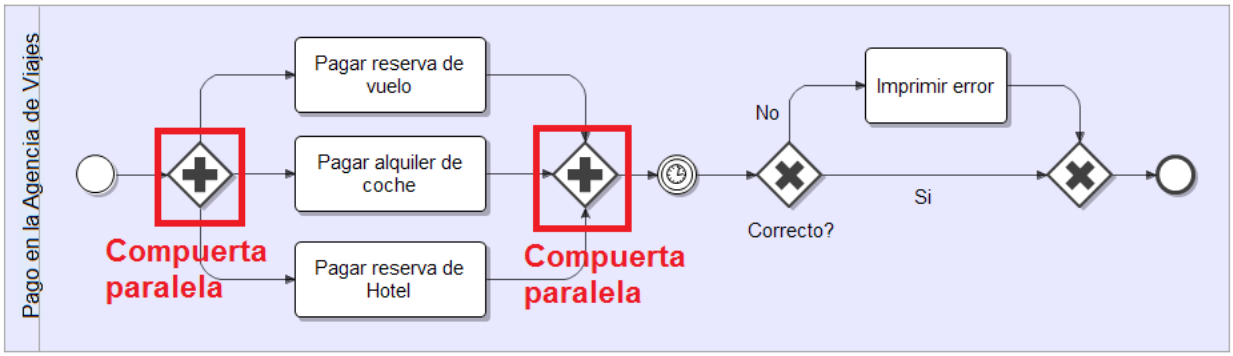

Figura 12: Diagrama BPMN que describe los pagos internos en una agencia de viajes con el remarcado de las compuertas paralelas

Objetos de conexión

Los *Objetos de Conexión* permiten conectar cada uno de los objetos de conexión. Podemos encontrarlos de tres tipos: *Secuencias*, *Mensajes* y *Asociaciones*. De estos tres, el único soportado por la transformación BABEL es el de secuencia.

El *Flujo de secuencia* está representado por una línea simple continua y flechada y muestra el orden en que las actividades se llevarán a cabo, tal como se puede ver en la Figura 13.

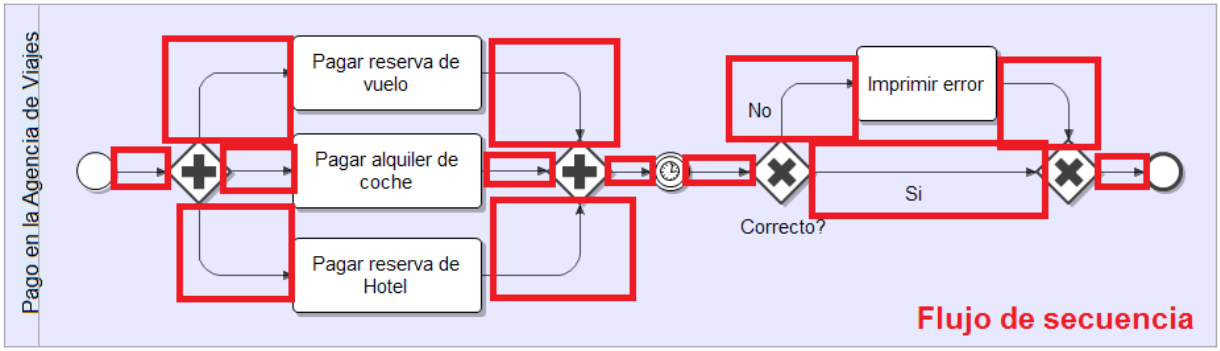

Figura 13: Diagrama BPMN que describe los pagos internos en una agencia de viajes con el remarcado de los flujos de secuencia

2.1.5. Business Process Execution Language (WS-BPEL)

El Web Service Business Process Execution Language (WS-BPEL) es un estándar de OASIS, el cual se desarrolló a partir de WSFL y XLANG, ambos lenguajes basados en XML y orientado a la descripción de servicios Web. Básicamente, WS-BPEL consiste en un lenguaje basado en XML diseñado para el control centralizado de la invocación de diferentes servicios Web, con cierta lógica de negocio añadida que ayudan a la programación a gran escala.

Tecnologías

WS-BPEL incluye en su especificación referencias a numerosos estándares de servicios Web y XML. Algunos de los más importantes son:

- WSDL es el más importante de estos estándares porque WS-BPEL describe los procesos de negocios como conversaciones entre servicios Web, los cuales se describen en WSDLs. Además, los procesos WS-BPEL son servicios Web, por lo que también utilizan WSDL para describir su interfaz abstracta y la forma en la que el cliente se comunicará con el proceso para consumirlo.

- XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así una percepción del tipo de documento con un nivel alto de abstracción. XML Schema es utilizado para especificar tipos (por ejemplo, int, doublé, datosDelVuelo) dentro del proceso WS-BPEL.

- XPath (XML Path Language) es un lenguaje que permite construir expresiones que recorren y procesan un documento XML. XPATH proporciona al proceso WS-BPEL una forma de manejar sus datos.

Estructura de un proceso de negocio WS-BPEL

Un proceso WS-BPEL es un contenedor donde se incluye la declaración de las relaciones con los servicios Web externos, la declaración de los datos del proceso, manejadores para distintos propósitos y lo más importante, las actividades a ser ejecutadas.

Básicamente WS-BPEL construye procesos de negocios a partir de servicios Web. Es un lenguaje que construye y extiende los servicios Web, armando nuevos servicios compuestos.

La definición de un proceso WS-BPEL consta de dos tipos de archivos: 1) archivos WSDL que especifican las interfaces de los servicios Web que intervienen en el proceso y la interfaz del propio proceso; y 2) un archivo WS-BPEL que es un documento XML con la definición de un proceso con las actividades principales, enlaces, variables, etc.

Para clarificar la estructura del proceso, se utiliza un ejemplo para guiar la explicación (ver la Figura 14). Este proceso es una sección del caso de estudio (descrito en el capítulo 5). Este proceso recibe un conjunto de parámetros con los que se invoca a un servicio Web, el cual busca todos los seguros de viajes relevantes en función de los datos de entrada. Si el servicio Web retorna un error, se repite la llamada a éste. En caso contrario, se retorna el resultado al usuario.

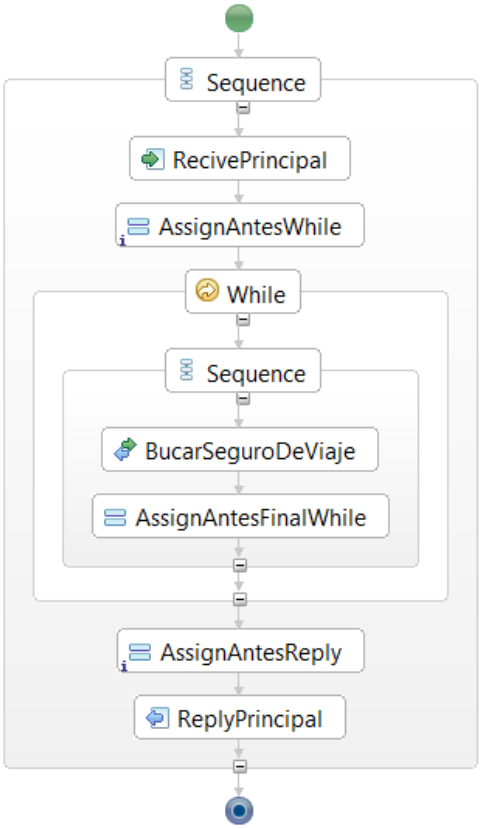

Figura 14: Proceso de búsqueda de seguros de viaje

La Figura 15 es la representación textual del proceso WS-BPEL.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasis-
open.org/wsbpel/2.0/process/executable"
xmlns:p="http://www.w3.org/2001/XMLSchema"xmlns:tns="http://BABEL_SearchInsurance"
xmlns:wsdl3="http://www.example.org/KingInsurance/"
xmlns:xsd2="http://www.example.org/XMLSchemaInsurance"
name="BABEL_SearchInsurance" targetNamespace="http://BABEL_SearchInsurance">

  <!--Imports-->
  <bpel:import location="BABEL_SearchInsurance.wsdl"
namespace="http://BABEL_SearchInsurance"
importType="http://schemas.xmlsoap.org/wsdl/" />

  <!--PartnerLinks-->
  <bpel:partnerLinks>
    <bpel:partnerLink name="client" partnerLinkType="tns:BusquedaDeSegurosPLT"
myRole="BusquedaDeSegurosProvider" />
    <bpel:partnerLink name="KingInsurancePartnerLink"
partnerLinkType="tns:PLTKingInsurance" partnerRole="RoleKingInsurance" />
  </bpel:partnerLinks>

  <!--Variables-->
  <bpel:variables>
    <bpel:variable name="EntradaProcesoVariable"
messageType="tns:EntradaBusquedaDeSegurosMessage" />
    <bpel:variable name="SalidaProcesoVariable"
messageType="tns:SalidaBusquedaDeSeguroMessage" />
    <bpel:variable name="EntradaBusquedaSeguroVariable"
messageType="wsdl3:searchInsuranceRequest" />
    <bpel:variable name="SalidaBusquedaSeguroVariable"
messageType="wsdl3:searchInsuranceResponse" />
    <bpel:variable name="error" type="p:boolean" />
  </bpel:variables>

  <!--Main Sequence-->
  <bpel:sequence>
    <bpel:receive name="RecivePrincipal" partnerLink="client"
portType="tns:BusquedaDeSegurosPortType" operation=" BusquedaSegurosDeViajeBPEL"
variable="EntradaProcesoVariable" createInstance="yes" />
    <bpel:assign name="AssignAntesWhile">
      <bpel:copy>
        <bpel:from>
          <bpel:literal>
            <tns:RequestSearchInsurance
xmlns:tns="http://www.example.org/XMLSchemaInsurance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />
          </bpel:literal>
        </bpel:from>
        <bpel:to>$EntradaBusquedaSeguroVariable.parameters</bpel:to>
      </bpel:copy>
      <bpel:copy>
        <bpel:from>true()</bpel:from>
        <bpel:to>$error</bpel:to>
      </bpel:copy>
    </bpel:assign>
    <bpel:while name="While">
      <bpel:condition>$error</bpel:condition>
```

```
58        <bpel:sequence>
59          <bpel:invoke name="BucarSeguroDeViaje"
60  partnerLink="KingInsurancePartnerLink" portType="wsdl3:KingInsurance"
61  operation="searchInsurance" inputVariable="EntradaBusquedaSeguroVariable"
62  outputVariable="SalidaBusquedaSeguroVariable" />
63          <bpel:assign name="AssignAntesFinalWhile">
64            <bpel:copy>
65
66  <bpel:from>$SalidaBusquedaSeguroVariable.parameters/xsd2:error</bpel:from>
67              <bpel:to>$error</bpel:to>
68            </bpel:copy>
69          </bpel:assign>
70        </bpel:sequence>
71      </bpel:while>
72      <bpel:assign name="AssignAntesReply">
73        <bpel:copy>
74          <bpel:from>
75            <bpel:literal>
76              <tns:ResponseSearchInsurance
77  xmlns:tns="http://www.example.org/XMLSchemaInsurance"
78  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
79                <tns:inurances>
80                  <tns:idInsurance>0</tns:idInsurance>
81                  <tns:name>tns:name</tns:name>
82                </tns:inurances>
83                <tns:error>true</tns:error>
84                <tns:errorCode>0</tns:errorCode>
85                <tns:errorExplanation>tns:errorExplanation</tns:errorExplanation>
86              </tns:ResponseSearchInsurance>
87            </bpel:literal>
88          </bpel:from>
89          <bpel:to>$SalidaBusquedaSeguroVariable.parameters</bpel:to>
100       </bpel:copy>
101       <bpel:copy>
102        <bpel:from>$SalidaBusquedaSeguroVariable.parameters</bpel:from>
103        <bpel:to>$SalidaProcesoVariable.parameters</bpel:to>
104       </bpel:copy>
105     </bpel:assign>
106     <bpel:reply name="ReplyPrincipal" partnerLink="client"
107 portType="tns:BusquedaDeSegurosPortType" operation=" BusquedaSegurosDeViajeBPEL"
108 variable="SalidaProcesoVariable" />
109   </bpel:sequence>
110 </bpel:process>
```

Figura 15: Código del proceso de búsqueda de seguros de viaje en WS-BPEL

En la Figura 15, se pueden diferenciar cuatro elementos principales, los cuales son comunes en todo proceso WS-BPEL:

Proceso

El proceso define el nombre y el espacio de nombres. Un espacio de nombres es un conjunto de nombres en el cual todos son únicos. A cada nombre se le añade una abreviatura que permite referenciarlo a lo largo del proceso. Además, el elementos proceso engloba al resto de elementos. En el ejemplo, su definición se encuentra entre las líneas 2-7.

PartnerLinks

Los *PartnerLinks* definen los servicios u otros procesos con los cuales va a interactuar el proceso en cuestión. En la Figura 15 se han definido dos *PartnerLinks*.

1) El primero, que se encuentra entre las líneas 16-17, representa al propio proceso, ya que éste no es más que otro servicio Web, que interactúa con otros servicios Web.

2) El segundo, que se encuentra entre las líneas 18-19, permite interactuar con el servicio Web, que busca los seguros de viajes con las características deseadas.

Además, todo *PartnerLink* tiene que especificar su *PartnerLinkType*. Éste se encarga de especificar la relación entre dos servicios, mediante la definición de los roles jugados en la conversación y mediante la especificación de los *PortTypes* proporcionados por cada uno de los servicios. La definición de los dos *PartnerLinkTypes* se encuentra en la Figura 16 entre las líneas 40-46.

Variables

Las *Variables* permiten almacenar información dentro de un proceso. En la Figura 15 las variables se encuentran definidas entre las líneas 23-33. En este proceso, se pueden encontrar dos tipos de variables:

1) Las que son de tipo mensaje que se utilizan como entrada y salida de las diferentes invocaciones a servicios. Un ejemplo de este tipo es la variable con nombre "SalidaBusquedaSeguroVariable", la cual se usa para almacenar el resultado de la búsqueda de seguros.

2) Las que son de tipo simple se suelen utilizar para la implementación de la lógica dentro del proceso. En este caso solo hay una variable de este tipo con el nombre "error", y se utiliza para implementar la condición del bucle.

Cuerpo

El cuerpo del proceso es típicamente una actividad, compuesta por otras actividades. Es en el cuerpo donde se organiza como se va a interactuar con el resto de servicios, y la lógica que hay detrás de ello.

Además, como se ha mencionado al inicio de este apartado, todo proceso WS-BPEL está formado por dos elementos, el documento WS-BPEL el cual se puede ver en la Figura 15 y el WSDL del proceso que se muestra en la Figura 16.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
3  xmlns:p="http://www.w3.org/2001/XMLSchema" xmlns:plnk="http://docs.oasis-
4  open.org/wsbpel/2.0/plnktype"
5  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
xmlns:tns="http://BABEL_SearchInsurance"
xmlns:wsdl3="http://www.example.org/KingInsurance/"
xmlns:xsd2="http://www.example.org/XMLSchemaInsurance"
name="BABEL_SearchInsurance" targetNamespace="http://BABEL_SearchInsurance">

  <!-- Definition Import -->
    <wsdl:import namespace="http://www.example.org/KingInsurance/"
location="KingInsurance.wsdl"></wsdl:import>

<!-- Definition Types -->
  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import schemaLocation="XMLSchemaInsurance.xsd"
namespace="http://www.example.org/XMLSchemaInsurance" />
    </xsd:schema>
  </wsdl:types>

  <!-- Definition Messages -->
  <wsdl:message name="SalidaBusquedaDeSeguroMessage">
    <wsdl:part element="xsd2:SalidaBusquedaDeSeguros" name="parameters" />
  </wsdl:message>
  <wsdl:message name="EntradaBusquedaDeSegurosMessage">
    <wsdl:part element="xsd2:EntradaBusquedaDeSeguros" name="parameters" />
  </wsdl:message>

  <!-- Definition PortType -->
  <wsdl:portType name="BusquedaDeSegurosPortType">
    <wsdl:operation name="BusquedaSegurosDeViajeBPEL">
      <wsdl:input message="tns:EntradaBusquedaDeSegurosMessage" />
      <wsdl:output message="tns:SalidaBusquedaDeSeguroMessage" />
    </wsdl:operation>
  </wsdl:portType>

  <!-- Definition PartnerLinkTypes -->
  <plnk:partnerLinkType name="BusquedaDeSegurosPLT">
    <plnk:role name="BusquedaDeSegurosProvider"
portType="tns:BusquedaDeSegurosPortType" />
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name="PLTKingInsurance">
    <plnk:role name="RoleKingInsurance" portType="wsdl3:KingInsurance" />
  </plnk:partnerLinkType>

  <!-- Definition Binding -->
  <wsdl:binding name="BusquedaDeSegurosBinding"
type="tns:BusquedaDeSegurosPortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="BusquedaSegurosDeVaijesBPEL">
      <soap:operation
soapAction="http://BABEL_SearchInsurance/SearchInduranceBPEL" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

```

```
65    <!-- Definition Service -->
66    <wsdl:service name="BusquedaDeSeguros">
67      <wsdl:port name="BusquedaDeSegurosPort"
68  binding="tns:BusquedaDeSegurosBinding">
69        <soap:address
70  location="http://localhost:8081/ode/processes/BABEL_SearchInsurance" />
71      </wsdl:port>
72    </wsdl:service>
73   </wsdl:definitions>
74
```

Figura 16: WSDL del proceso inicial

En la Figura 16, se puede ver cómo se han definido dos mensajes "SalidaBusquedaDeSeguroMessage" y "EntradaBusquedaDeSegurosMessage" entre las líneas 24-29. Estos mensajes definen la entrada y la salida del proceso principal, el cual se ha definido con una única operación de nombre "BusquedaSegurosDeViajeBPEL" cuya definición se encuentra entre las líneas 33-36.

Actividades de WS-BPEL

A continuación se explican las diferentes actividades que pueden ser encontradas dentro del cuerpo del proceso WS-BPEL. Éstas se pueden clasificar en dos grupos: 1) Actividades básicas que desarrollan el cometido para el que fueron creadas, como es recibir un mensaje, manipular datos, etc.; y 2) Actividades estructuradas que contienen a otras actividades y la lógica de negocio entre ellas.

En la Figura 17 se ve cada una de las actividades y sus respectivo grupo.

Receive	Actividades básicas
Reply	
Invoke	
Assign	
Wait	
Sequence	Actividades estructuradas
Flow	
If	
Switch	
While	
RepeatUntil	
Pick	

Figura 17: Actividades en WS-BPEL

Para la explicación de cada una de las actividades se utiliza el ejemplo de la búsqueda de seguros (en los casos en los que esta actividad aparezca). En los casos en los que la actividad no ha sido utilizada en el ejemplo, ésta es explicada mediante su especificación formal, la cual puede ser encontrada en el estándar WS-BPEL 2.0 [**5**].

Receive

Esta actividad espera por un mensaje que llegue de un cliente que pretende hacer uso del proceso WS-BPEL. Debe especificar el nombre del *PartnerLink* del cual espera que llegue el mensaje, el *PortType* y la operación que desea invocar. Adicionalmente puede especificarse la variable por la cual recibirá el mensaje.

Entre las líneas 37-39 en el ejemplo de la Figura 15 se puede ver un ejemplo de uso del *Receive*. Éste es utilizado para recibir los parámetros necesarios para la ejecución de la búsqueda de seguros. Estos parámetros son almacenados en la variable especificada y posteriormente son utilizados para la invocación del servicio Web de la búsqueda de seguros.

Reply

Esta actividad permite enviar un mensaje en respuesta a otro mensaje recibido a través de una actividad *Receive*. Al igual que la actividad *Receive*, debe especificar el *PartnerLink*, *PortType* y *Operation*, al cual hace referencia.

En el ejemplo de la Figura 15 entre las líneas 106-108, se puede ver un ejemplo de uso del *Reply*. Éste retorna el resultado de la ejecución del proceso y lo finaliza.

Invoke

Esta actividad permite invocar a un servicio Web. Al igual que con la actividad de tipo *Reply* y *Receive*, se deben especificar el *PartnerLink*, *PortType* y *Operation* del servicio Web que se desea invocar. Adicionalmente se puede indicar la variable que se le pasa como entrada al servicio Web y la variable donde se almacenará el mensaje de respuesta.

En el ejemplo de la Figura 15 entre las líneas 59-62, se puede ver un ejemplo de uso del *Invoke*. Éste es utilizado para la invocación del servicio Web de la búsqueda de seguros. Para su invocación se utilizan dos variables, las cuales están definidas entre las líneas 28-32, que permiten especificar los datos de entrada y almacenar el resultado de la invocación.

Assign

Esta actividad es el principal elemento para la manipulación de datos en WS-BPEL y toma como base el lenguaje XPath estándar, a través de sus consultas, expresiones y funciones. Además, es posible usar XQuery, XSLT o Java para manipulaciones o transformaciones más complejas de datos.

En el ejemplo de la Figura 15 , se han utilizado tres actividades de este tipo.

1) Dos de ellas, las cuales se encuentran entre las líneas 40-55 y 72-105, tienen la función de mapear la variable de entrada del proceso con la variable de entrada a la

invocación del servicio Web y la variable de salida de la invocación con la variable de salida del proceso respectivamente.

2) El *Assign* restante, que se encuentra entre las líneas 63-69, se utiliza para dar valor a la variable de nombre "error" que es utilizada como condición del bucle.

En general, como se ha podido ver en los ejemplos, toda actividad *Assign* está formada por dos componentes *From* y *To*, los cuales son utilizados para indicar el origen y el destino del mapeo.

Wait

Esta actividad permite esperar un cierto periodo de tiempo. La manera de especificar las expresiones asociadas a los atributos *for* y *until* se logran mediante expresiones en XPath 1.0. Su sintaxis formal se puede observar en la Figura 18.

```
1  <wait (for="duration-expr" | until="deadline-expr") >
2  </wait>
```

Figura 18: Sintaxis formal de la actividad Wait en WS-BPEL

Sequence

Una *Sequence* contiene una o más actividades que se ejecutan secuencialmente en el orden en que fueron listadas. Esta actividad finaliza cuando termina la última actividad de la misma.

En el ejemplo de la Figura 15, se han utilizado dos actividades de este tipo.

1) La primera, que se encuentra entre las líneas 36-109, engloba a todo el resto de actividades, y como se ha explicado, indica que las actividades que contiene se ejecutarán de manera secuencial.

2) La segunda, que se encuentra entre las línea 58-70, engloba todas las actividades dentro de la actividad *While*. Al igual que en el caso anterior, marca la ejecución secuencial de las actividades que contiene.

En general, el uso de la actividad *Sequence*, al igual que la actividad *Flow* está marcado por restricciones de otras actividades. Por ejemplo, el cuerpo de un proceso sólo puede estar formado por una actividad, por lo que ésta deberá ser una actividad que pueda englobar al resto. Lo mismo ocurre en el caso del *While*.

Flow

Esta estructura permite que dos o más actividades sean ejecutadas concurrentemente y concluye cuando todas las actividades que engloba hayan finalizado. Su sintaxis formal se puede observar en la Figura 19.

```
<flow>
      Activity*
</flow>
```

Figura 19: Sintaxis formal de la actividad Flow en WS-BPEL

If

Esta actividad proporciona un comportamiento condicional al proceso (al estilo de un if en Java). La actividad consiste en una lista de una o más condiciones (en forma de expresiones Xpath), que permite seleccionar una rama de ejecución entre varias. Su sintaxis formal se puede observar en la Figura 20.

```
<if>
   <condition>bool-expr</condition>
   activity
   <elseif>+
      <condition>bool-expr</condition>
      activity
   </elseif>
   <else>?
      activity
   </else>
</if>
```

Figura 20: Sintaxis formal de la actividad IF en WS-BPEL

Switch

Este tipo de actividad cumple la función de una estructura de selección múltiple (al estilo del *case* de Java) en donde es posible introducir condiciones (en forma de expresiones XPath) que permitan seleccionar una rama de ejecución entre varias. Su sintaxis formal se puede observar en la Figura 21.

```
<switch>
      <case condition="bool-expr">+
            activity
      </case>
      <otherwise>?
            activity
      </otherwise>
</switch>
```

Figura 21: Sintaxis formal de la actividad Switch en WS-BPEL

While

El *While* permite especificar que una actividad se ejecutará hasta que se alcance cierta condición. En las líneas 56-71 de la Figura 15 se puede ver un ejemplo de uso del *While*. Éste se encarga de repetir la invocación del servicio Web de búsqueda de seguros mientras el resultado contenga algún error.

RepeatUntil

El comportamiento del *RepeatUntil* es muy similar al *While*, con la diferencia que la actividad que contiene se ejecutará como mínimo una vez. Su sintaxis formal se puede observar en la Figura 22.

```
<repeatUntil >
   activity
   <condition> bool-expr</condition>
</repeatUntil>
```

Figura 22: Sintaxis formal de la actividad RepeatUntil en WS-BPEL

Pick

La actividad *Pick* está formada por un conjunto de ramas de la forma evento/actividad. Estas ramas pueden ser de dos tipos:

- *onAlarm*: sigue un comportamiento idéntico al de una actividad *Wait*, indicando mediante una expresión Xpath cuando se activará.
- *onMessage*: sigue un comportamiento idéntico al de una actividad *Receive*. Esta rama se activará cuando llegue el mensaje esperado.

Cuando ocurre alguno de estos desencadenadores, se ejecuta la actividad asociada a cada uno de ellos. Cada actividad *Pick* debe incluir por lo menos un desencadenador *onMessage*. La sintaxis formal se puede observar en la Figura 23.

```
<pick createInstance="yes|no"? >
      <onMessage partnerLink="ncname" portType="qname"
        operation="ncname" variable="ncname"?>+
            activity
      </onMessage>
      <onAlarm (for="duration-expr" | until="deadline-expr")>*
            activity
      </onAlarm>
</pick>
```

Figura 23: Sintaxis formal de la actividad Pick en WS-BPEL

2.2. Desarrollo de la propuesta

Dada la complejidad y amplitud de WS-BPEL, y debido a las necesidades específicas de los procesos de negocio y de las empresas actuales, se hace necesario el desarrollo de una herramienta eficiente para el desarrollo y mantenimiento de los procesos de negocio.

Como se ha comentado en el Capítulo 2, para la implementación de esta herramienta, se propone la implementación de la transformación de BPMN a WS-BPEL, y la construcción de una interfaz gráfica, que permita completar el WS-BPEL haciéndolo

totalmente operativo. La herramienta propuesta es atractiva, sencilla para los usuarios, y altamente abstracta gracias a su orientación basada en modelos.

2.3. Tecnologías utilizadas

En este apartado se tratan las distintas tecnologías que se han utilizado para el desarrollo de la propuesta. Ésta ha sido desarrollada exclusivamente con software libre, tratando de aprovechar al máximo la gran cantidad de herramientas disponibles.

2.3.1. Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria [**35**].

Las características principales de Java son las siguientes:

- La orientación a objetos se basa en diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables

- La independencia de plataforma permite que programas escritos en el lenguaje Java puedan ejecutarse igualmente en cualquier tipo de hardware.

- El recolector de basura simplifica la tarea del programador. El programador determina cuándo se crean los objetos y es el entorno en tiempo de ejecución el responsable de gestionar el ciclo de vida de los objetos. El programa u otros objetos pueden tener localizado un objeto mediante una referencia a éste. Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto.

2.3.2. Eclipse

Eclipse es un entorno de desarrollo integrado (IDE, Integrated Development Environment) de código abierto, multiplataforma para desarrollar software [**36**]. Eclipse fue diseñado para facilitar la edición, compilación y ejecución de programas, durante la fase de desarrollo. Eclipse es una aplicación gratuita y de código abierto disponible en la red para su descarga [**37**]. Para la implementación de la propuesta se ha utilizado la versión de Eclipse Indigo 3.7.3.

2.3.3. JDOM

En el campo de la manipulación de documentos XML, se pueden encontrar numerosas alternativas compatibles con Java, tales como XAM, DOM, JDOM y XOM. A la vista de una gran cantidad de alternativas, se evaluaron las diferentes posibilidades, buscando la simplicidad de la implementación y la eficiencia (tiempo usado para la ejecución de la transformación).

Algunos de los motivos que nos han hecho elegir JDOM [**38**], en lugar de las otras alternativas, han sido: 1) permite realizar las operaciones sobre el XML de una manera simple e intuitiva; 2) mayor integración con XPATH. En la propia API se proporcionan mecanismos para la ejecución de sentencias XPATH; y 3) a diferencia de DOM, JDOM ha sido creado y optimizado específicamente para Java. Esto hace que sea una API más eficiente y más natural de usar para el desarrollador Java y por lo tanto requiere un menor coste de aprendizaje.

JDOM es una API desarrollada específicamente para Java que da soporte al tratamiento de XML: parseo, búsquedas, modificación, generación y serialización. Éste es un modelo similar a DOM, pero no está creado ni modelado sobre DOM. Se trata de un modelo alternativo [**38**].

Concretamente, se ha utilizado la versión de JDOM 2.0.1 que ha sido significativamente restructurada con respecto a su anterior versión mediante el uso de genéricos y otras características introducidas en la versión de java 5. Esta API junto al lenguaje XPATH, del cual se habla en el siguiente apartado, han sido los componentes claves para la implementación de las transformaciones de BPMN a WS-BPEL.

2.3.4. XPATH

XPath (*XML Path Language*) es un lenguaje que permite construir expresiones que recorren y procesan un documento XML. La idea es parecida a las expresiones regulares para seleccionar partes de un texto sin atributos (*plain text*) [**39**]. XPath proporciona el dinamismo que se requería para la propuesta, y es el elemento principal responsable de la recuperación y filtrado de información de los documentos XML. A continuación, se muestra un ejemplo de la sintaxis de XPath, en donde se puede ver este dinamismo: "*//invoke[@name='Buscar_seguros_de_viaje']*". Esta sentencia retorna todos los nodos de un documento XML con nombre "invoke" y con un atributo "name" igual a "*Buscar_seguros_de_viaje*".

Una operación tan sencilla como esta, que consta de una línea de código con XPath, implica sin XPath, una gran cantidad de líneas de código. Esto se debe principalmente al tipo de documento con el que estamos trabajando. El XML que representa un proceso WS-BPEL organiza sus elementos en función de su posición dentro del proceso (no por tipo del

elemento). Por lo que un *Invoke* puede estar dentro de diferentes elementos, por ejemplo un *While*, un *Sequence* entre otros. Es por esto que recorrer de forma secuencial el documento buscando un elemento sin XPath es muy costoso y engorroso. Además, utilizar este tipo de búsqueda relativa permite un resultado mas robusto frente a modificaciones en la organización del documento, haciendo así, una solución mas perdurable en el tiempo.

2.3.5. STP BPMN

STP BPMN es un subproyecto de SOA Tools Platform en Eclipse [**40**]. Su principal objetivo es la construcción de un marco de trabajo y un conjunto de herramientas extensibles, que permiten el diseño, configuración, montaje, despliegue, monitorización y gestión del software, diseñado alrededor de una SOA.

Dentro de este marco, SOA Tools Platform está formado por diversos subproyectos. STP BPMN es uno de sus subproyectos activos actualmente, el cual, ha sido usado para la elaboración de la presente propuesta. STP BPMN proporciona un editor y un conjunto de herramientas para la creación de diagramas de procesos de negocio basados en la notación BPMN (ver sección 3.2.1).

La Figura 24 muestra el editor proporcionado por el proyecto STP BPMN. En éste se diferencian claramente dos secciones: en la sección A se modela el proceso y se insertan los diferentes elementos que contiene un diagrama BPMN. En la sección B se muestra la paleta del editor, la cual permite ver y seleccionar todos los elementos BPMN soportados por la herramienta.

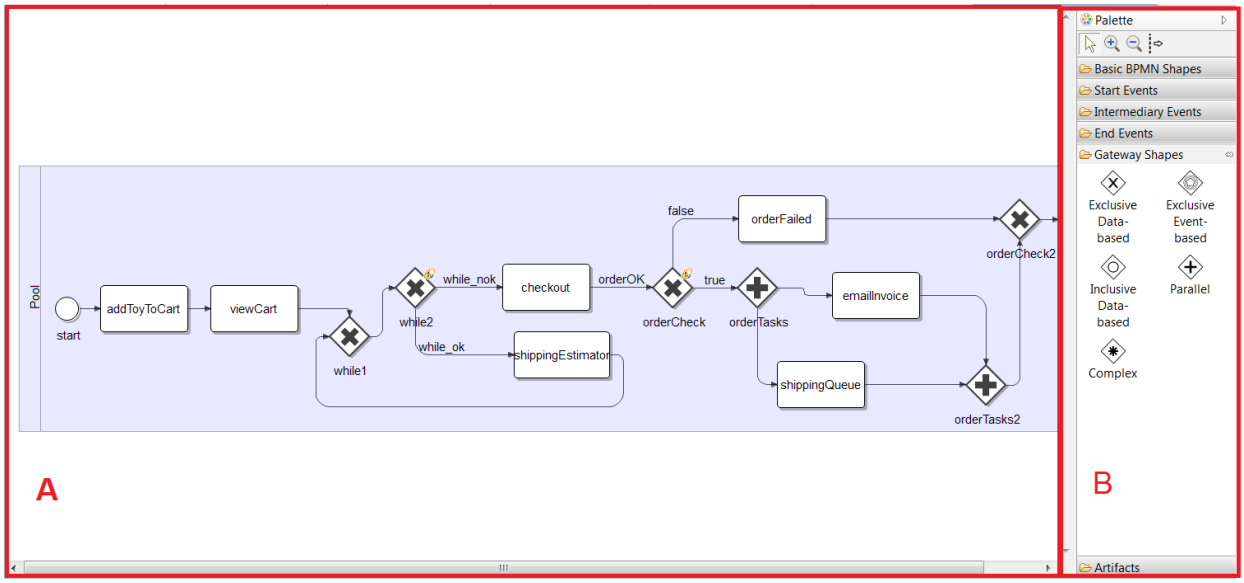

Figura 24: Editor de Diagramas BPMN en STP BPMN

2.3.6. BPEL Designer Project

BPEL Designer Project es un proyecto de Eclipse que proporciona soporte completo para la definición, creación, edición, implementación, pruebas y depuración de procesos WS-BPEL bajo el estándar WS-BPEL 2.0 [**5**,**41**]. Esta herramienta se puede extender por terceros y permite la manipulación de procesos BPEL de una manera gráfica intuitiva.

BPEL Designer ha desarrollado su propia representación gráfica de los diferentes elementos que componen un proceso WS-BPEL.

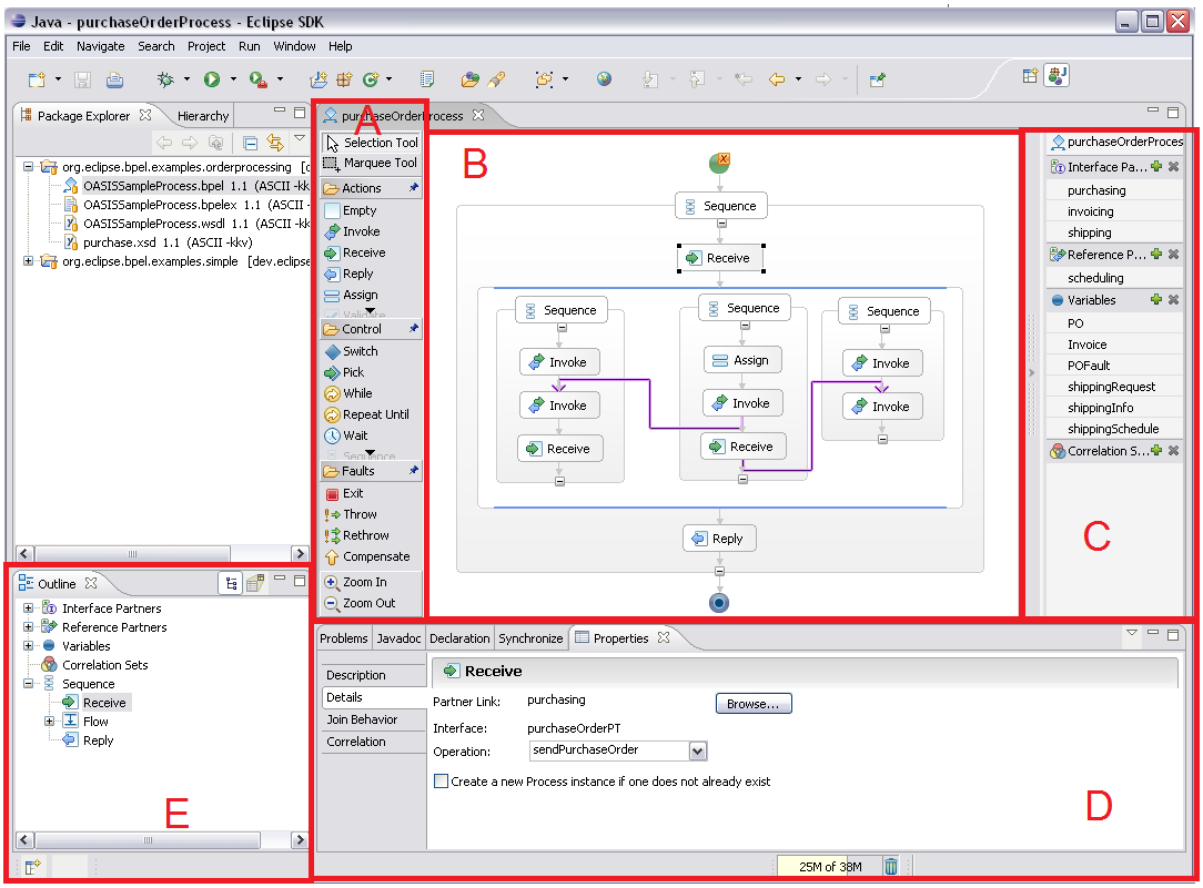

Figura 25: Editor de procesos BPEL en BPEL Designer

La Figura 25 muestra el editor visual que proporciona BPEL Designer. En él, se pueden diferenciar las siguientes secciones: A) es la paleta del editor, en ella aparecen todos los elementos BPEL disponibles para insertar en el proceso; B) es la sección principal del editor que permite la edición del proceso mediante la inserción de nuevos elementos y mediante la especificación del orden de estos elementos; C) muestra los elementos inherentes al proceso principal, los cuales no tienen representación visual; D) muestra las propiedades del elemento seleccionado actualmente; y E) permite ver el proceso en forma de workflow.

2.3.7. Apache ODE

Apache ODE (Orchestration Director Engine) es una herramienta desarrollada por Apache Software Fundation bajo la licencia Apache, que permite la ejecución de procesos de negocio ejecutables basados en el estándar WS-BPEL 2.0 [**5**,**11**].

Sus características principales son: 1) soporte del estándar WS-BPEL 2.0 de OASIS y el BPEL4WS 1.1; 2) soporte de ejecución bajo diferentes entornos; 3) soporte para HTTP WSDL Binding; 4) proporciona una API de alto nivel para acceso al motor que permite la integración del núcleo, con múltiples capas de comunicación e incluso su uso de manera independiente desde una aplicación; 5) la compilación de WS-BPEL provee de análisis detallados y validación desde la línea de comandos o durante el despliegue; y 6) interfaz para la gestión de procesos, instancias y mensajes.

La arquitectura de Apache ODE está dividida en diversos componentes, facilitando su reutilización. A continuación se muestran sus diferentes componentes:

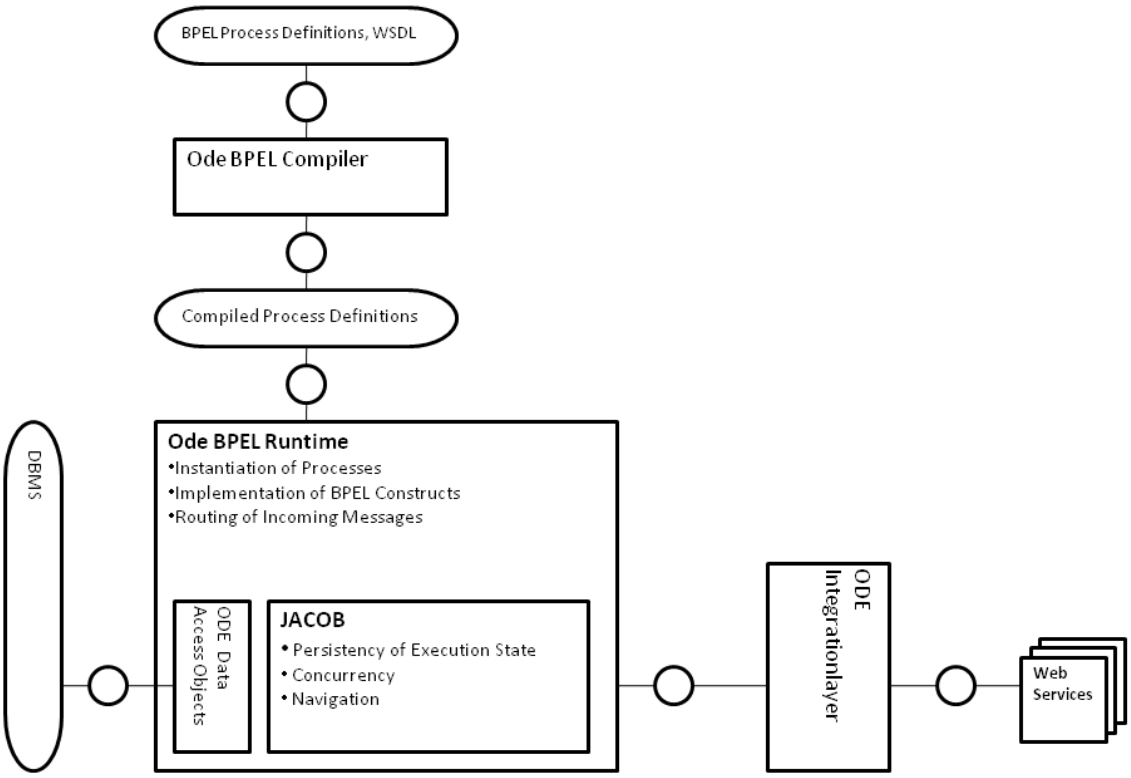

Figura 26: Arquitectura de Apache ODE [11]

En la Figura 26, se pueden ver los diferentes componentes que conforman la arquitectura de Apache ODE. A continuación se explican los más importantes:

- ODE BPEL Compiler: Es el componente encargado de la compilación de proceso WS-BPEL. Tiene como entrada el documento del proceso WS-BPEL, junto con sus

documentos asociados (WSDL, Schemas, etc.). Como salida, el proceso WS-BPEL compilado.

- ODE BPEL Runtime: Es el componente encargado de la ejecución del proceso WS-BPEL compilado.
- ODE Integration Layer: Permite al componente *ODE BPEL Runtime* comunicarse con el mundo exterior.

Para el desarrollo de la propuesta se ha utilizado Apache ODE 1.3.5 desplegado como un WAR en un servidor de aplicaciones Tomcat 6.0. Aunque actualmente ya está disponible la versión 7.0, ésta aun no era estable al inicio del desarrollo de la propuesta.

2.3.8. EMF project

EMF (Eclipse Modeling Framework) es un marco de trabajo de modelado y de generación de código para construir herramientas y otras aplicaciones basadas en un modelo de datos estructurado. Desde una especificación del modelo descrito en XMI, EMF suministra herramientas y soporte en tiempo de ejecución para producir un conjunto de clases Java para el modelo, un conjunto de clases Adapter que permiten la visualización y edición con base en comandos del modelo, y un editor básico [**42**].

Los modelos pueden ser especificados usando anotaciones en Java, documentos XML, o herramientas de modelado como Rational Rose [**43**], y después ser importados a EMF. Lo más importante es que EMF suministra las bases para el desarrollo dirigido por modelos.

2.3.9. FMF

FMF (Forms Modeling Framework) es un marco de trabajo que tiene como objetivo facilitar el desarrollo de editores Eclipse basados en formularios que manipulan modelos definidos mediante metamodelos Ecore [**44**,**45**].

Habitualmente, los modelos se editan mediante editores gráficos donde los elementos se representan en diagramas que utilizan figuras relacionadas entre ellas (como los modelos UML, BPMN, etc.). Trabajando sobre la plataforma Eclipse se puede utilizar el Graphical Modeling Framework (GMF) para desarrollar este tipo de editores. No obstante, en ocasiones estas metáforas gráficas no son las más adecuadas y resulta más sencillo e intuitivo editar el contenido de un modelo utilizando formularios (tablas, campos de texto, formularios maestro-detalle, etc.).

Utilizando FMF, se ha desarrollado una herramienta gráfica la cual permite al usuario modificar e introducir la información necesaria para la generación de un archivo WS-BPEL totalmente operativo.

2.3.10. BABEL Tool

BABEL es una herramienta desarrollada en la Queensland University of Technology para la generación de código WS-BPEL a partir de modelos BPMN. En las siguientes referencias, se puede encontrar información detallada de su funcionamiento e implementación [**15**,**46**].

Para la implementación de la herramienta, han creado su propio metamodelo BPMN y a partir de las instancias de éste, se genera el código WS-BPEL. En la Figura 27 se muestra el proceso interno seguido por BABEL para realizar la generación.

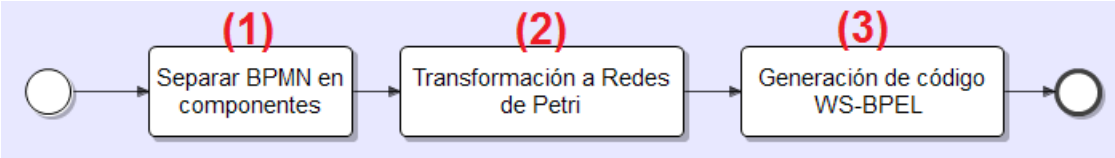

Figura 27: Implementación interna de la herramienta BABEL

Básicamente, el proceso implementado por la herramienta BABEL consta de tres transformaciones:

1) Separación del modelo BPMN en componentes: consiste en transformar la representación en forma de grafo del modelo BPMN en un representación estructurada en la que se separan componentes en función de la entrada y salida, es decir, se separa cada componente de forma que cada uno tenga una única entrada y salida.
2) Transformación en Redes de Petri: una Red de Petri es una representación matemática o gráfica de un sistema a eventos discretos en el cual se puede describir la topología de un sistema distribuido, paralelo o concurrente [**47**]. La transformación a un lenguaje formal como Redes de Petri permite evitar ambigüedades, simplificando la generación de código WS-BPEL final.
3) Generación de código WS-BPEL: a partir de unas reglas, se transforma cada uno de los componentes definidos en términos de Redes de Petri en su correspondiente sección de código WS-BPEL.

En el apartado 3.2 se explica con detalle el funcionamiento de BABEL a través de un ejemplo práctico.

2.3.11. Axis2

Axis2 es la versión mejorada del contenedor de servicios Web Axis. Este proyecto ha evolucionado independientemente de la primera versión debido a que implementa especificaciones diferentes [**48**,**49**].

Una de las principales características que han hecho que se escoja Axis2 para la implementación de los servicios Web no de terceros, es el amplio soporte que éste ofrece para los estándares relacionados con los servicios Web como son el WSDL 2.0, SOAP 1.2, y el WS-Adressing.

2.3.12. Apache Tomcat

Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) es un servidor Web con soporte de Servlets y Java Server Pages (JSPs) de Sun Microsystems. Fue desarrollado bajo el proyecto Jakarta en la Apache Software Foundation [**50**].

Tomcat es mantenido y desarrollado por miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de acceso libre a su código fuente y a su forma binaria, en los términos establecidos en la Apache Software Licence.

Tomcat puede funcionar como servidor Web por sí mismo. Hoy en día Tomcat es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Debido a que Tomcat está escrito en Java, éste funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Apache Tomcat ha sido el servidor escogido para la implementación de la propuesta. Éste ha sido utilizado para: 1) el despliegue de servicios Web utilizados en el caso de estudio; y 2) el despliegue y la ejecución de Apache ODE.

2.3.13. Hibernate

Hibernate es una herramienta de mapeo objeto-relacional para la plataforma Java (y disponible también para .Net con el nombre de NHibernate). Hibernate facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los *Beans* de las entidades, que permiten establecer estas relaciones [**51**].

Esta herramienta ha sido utilizada para la implementación de los servicios Web que no son ofrecidos por terceros, facilitando en gran medida la comunicación entre la base de datos y la capa lógica.

2.3.14. MySQL

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario, con más de seis millones de instalaciones. Al contrario de proyectos tales como Apache, donde el software es desarrollado por una comunidad pública y los derechos de autor del código están en poder del autor individual, MySQL es patrocinado por una empresa privada, que posee los derechos de autor de la mayor parte del código.

MySQL ha sido el gestor de base de datos utilizados para manejar las bases de datos que acceden los servicios Web que no son de terceros.

2.3.15. MySQL Workbench

MySQL Workbench es una herramienta que permite modelar diagramas de Entidad-Relación para bases de datos construidas en MySQL [**52**]. Con esta herramienta se puede elaborar una representación visual de las tablas, vistas, procedimientos almacenados y claves foráneas de la base de datos. MySQL Workbench es capaz de sincronizar el modelo en desarrollo con la base de datos real. Asimismo, esta herramienta apoya la realización de ingeniería inversa, para exportar e importar el esquema de una base de datos ya existente.

MySQL Workbench ha facilitado en gran medida la creación, actualización y mantenimiento de las diferentes bases de datos utilizadas en el desarrollo del caso de estudio.

Capítulo 3 Generación dirigida por modelos y despliegue en caliente de procesos WS-BPEL

En este capítulo se explican los pasos del proceso seguido desde la creación de un modelo BPMN que describe un proceso de negocios hasta la generación y despliegue de un proceso WS-BPEL completo. Para ello se han implementado una serie de transformaciones automáticas y semiautomáticas dirigidas por modelos que terminan con el despliegue en caliente del proceso WS-BPEL.

Este capítulo está dividido en cuatro secciones: 1) la primera proporciona la visión general del proceso implementado; 2) la segunda habla de BABEL y de la transformación de BPMN a WS-BPEL que éste implementa; 3) en tercer lugar se explica el proceso implementado para completar el WS-BPEL generado por BABEL; y 4) la última sección describe el despliegue en caliente del proceso, su realización y ventajas.

3.1. Descripción del proceso de transformación de BPMN a WS-BPEL

Este apartado proporciona una visión global de la propuesta implementada, las transformaciones y fases que la componen y el papel del usuario en cada una de ellas. En la Figura 28 se puede ver un modelo BPMN que representa el proceso implementado en esta propuesta. El proceso está dividido en los siguientes pasos:

1) Transformación de un modelo BPMN en un WS-BPEL inicial. Para ello se ha utilizado el proyecto BABEL, el cual proporciona una solución eficiente para la conversión de un modelo BPMN en un proceso WS-BPEL. No obstante, el proceso WS-BPEL generado por BABEL es incompleto, es necesario completarlo para que éste pueda ser desplegado y ejecutado.

2) Transformación automática del WS-BPEL inicial generado por BABEL en un modelo intermedio entendible por la herramienta gráfica utilizada para completar, de una forma sencilla para el usuario, la información faltante del proceso WS-BPEL generado por BABEL.

3) Edición por parte del usuario de un modelo intermedio a través de la herramienta implementada. El usuario introduce mediante la herramienta gráfica la información faltante, tal como WSDLs, XSDs, *PartnerLinks*, *Assigns*, etc.

4) Transformación de la información contenida en el modelo intermedio completado por el usuario y el WS-BPEL inicial en el documento WS-BPEL, el WSDL del proceso que define su interfaz abstracta y el fichero deploy.xml que proporciona la

localización final de los servicios Web. Al terminar esta transformación el proceso WS-BPEL esta completo, con los artefactos que lo componen, y listo para ser desplegado sobre el servidor.

5) En último lugar, para desplegar el proceso en el servidor, se crea una carpeta en éste, en la que se copian: el documento WS-BPEL, el WSDL del proceso, el fichero deploy.xml, y los WSDLs y XSDs que describen a los servicios Web que invoca el proceso a desplegar. Además, cada vez que se despliega el proceso, se crea una carpeta con un número de versión ascendente con el fin de no afectar las instancias de los procesos que actualmente se están ejecutando.

Con ello, queda desplegado y completamente operativo el proceso WS-BPEL, listo para su invocación y sin necesidad de reiniciar el servidor.

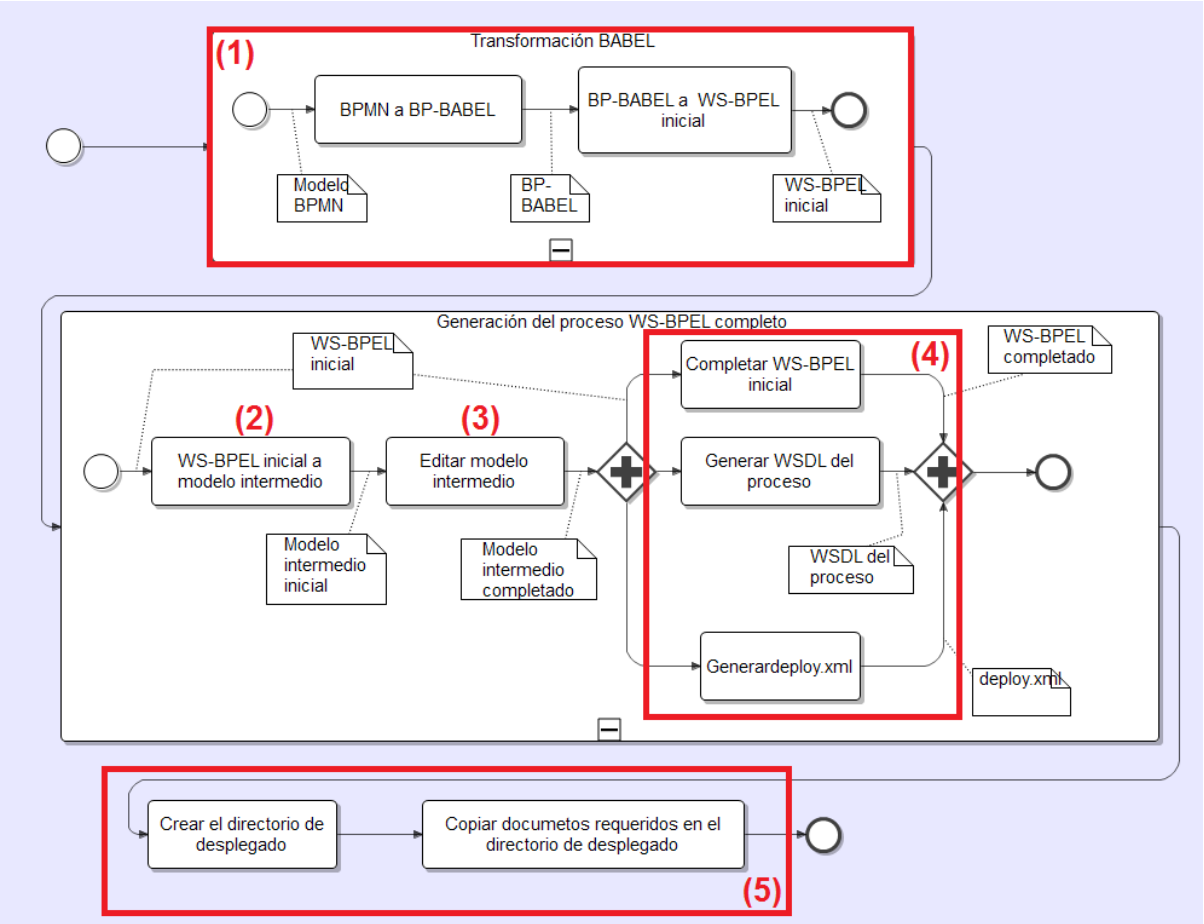

Figura 28: Proceso de transformación del modelo BPMN al proceso WS-BPEL que se despliega en caliente

A continuación se explican cada una de las transformaciones que conforman la propuesta. Para guiar la explicación de éstas, se ha elegido utilizar como ejemplo el proceso de reserva de actividades del caso de estudio descrito en el Capítulo 4. En la Figura 29 se puede ver el modelo BPMN que describe la lógica de este proceso compuesto por dos actividades: 1) Reservar actividad: se encarga de reservar una actividad (por ejemplo,

buceo, natación, paseo en barco, etc.) dado un hotel y el numero de personas que van a participar en esa actividad; y 2) Cancelar actividad: se encarga de cancelar la reserva de una actividad. Básicamente el usuario puede reservar tantas actividades como quiera, y en caso de producirse algún error, se cancelan todas las actividades que se hayan reservado hasta el momento.

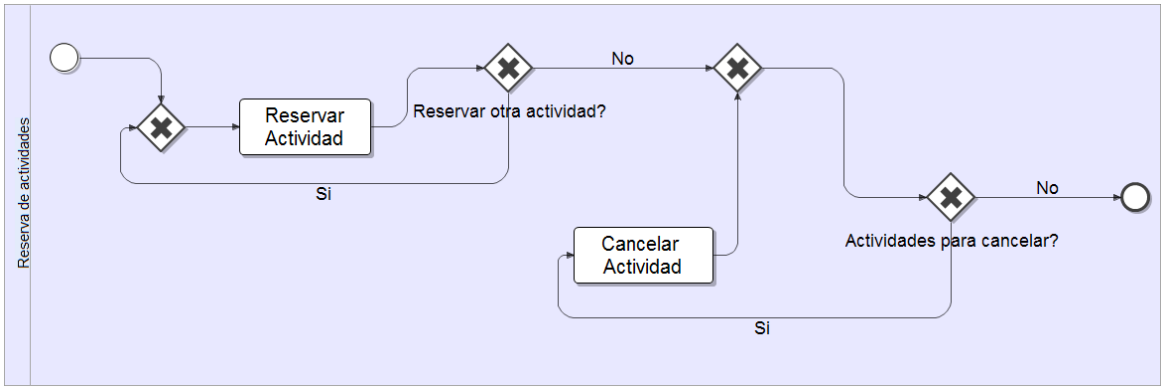

Figura 29: Reserva de actividades en BPMN

3.2. Trasformación de BPMN a un WS-BPEL inicial mediante BABEL

Para la generación del WS-BPEL inicial a partir de modelos BPMN se ha utilizado el proyecto BABEL [**15**]. Como se puede ver en la Figura 30, ésta consta de dos transformaciones.

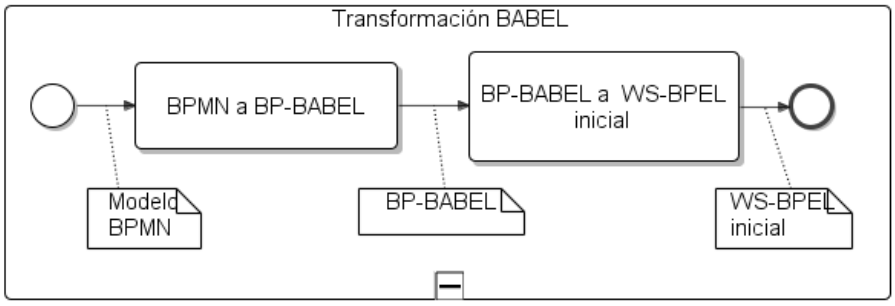

Figura 30: Transformación BABEL

La transformación del BPMN a BP-BABEL convierte el BPMN inicial en un proceso de negocio descrito de manera acorde al metamodelo de BABEL con el cual trabaja. Esta transformación modelo a modelo ha sido implementada mediante el Atlas Transformation Language (ATL) [**53**].

La transformación de BP-BABEL a WS-BPEL inicial convierte el proceso de negocio acorde al metamodelo BABEL en el código del proceso WS-BPEL. No obstante, el código generado es incompleto. Por ejemplo, falta la información relevante a los servicios Web

finales, así como la definición de las variables utilizadas en el WS-BPEL y los mapeos entres las diferentes variables y servicios que se invocan.

Ejemplo

Continuando con el ejemplo mostrado en el apartado 3.1, se ejecuta la transformación teniendo como entrada el BPMN mostrado en la Figura 29. Esta transformación convierte cada *Actividad* en el modelo BPMN en al menos en un *Invoke*, crea un *Receive* para iniciar el proceso y buscará estructuras de tipo *Switch* o de tipo *While* formadas por los *Puertas de Conexión*. En la Figura 31 se pueden ver los diferentes elementos de WS-BPEL encontrados en el diagrama BPMN.

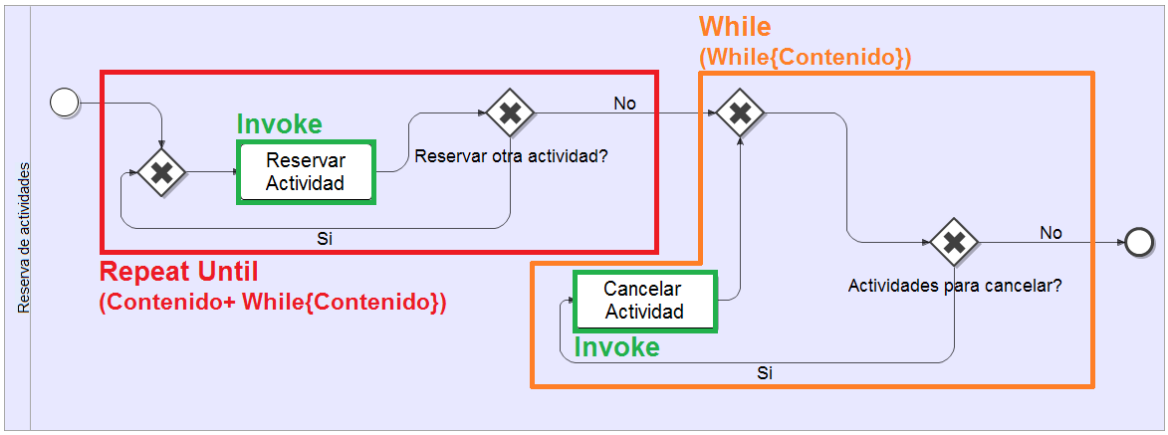

Figura 31: Reserva de actividades en BPMN con equivalencias en WS-BPEL

El código WS-BPEL que se obtiene después de la ejecución de la transformación BABEL se puede ver en la Figura 32. En ella, no están definidos los *Partner Links*, las *Variables*, los *Assigns*, etc. Este código es completado en las siguientes fases del proceso.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/abstract"
3  name="BABEL_BookActivities"
4  targetNamespace="http://samples.org//workspaceTransformation/STPbpmnTObabelBP
5  MN/models/BABEL_BookActivitiesBPMN.bpmn">
6    <!--bpmn2bpel Version 1.0-->
7    <partnerLinks>partnerLinks>
8    <variables><!--List variables used in this BPEL process--></variables>
9    <sequence name="sequenceComponent_2">
10     <receive name="ProcessInstantiation" partnerLink="client"
11 portType="localPT" operation="localPT" variable="client_data"
12 createInstance="yes" />
13     <sequence>
14       <invoke name="Reservar  Actividad" partnerLink="local"
15 portType="localPT" operation="Reservar  Actividad" inputVariable="Reservar
16 Actividad_data_in" outputVariable="Reservar  Actividad_data_out" />
17       <while condition="">
18         <invoke name="Reservar  Actividad" partnerLink="local"
19 portType="localPT" operation="Reservar  Actividad" inputVariable="Reservar
20 Actividad_data_in" outputVariable="Reservar  Actividad_data_out" />
21       </while>
```

```
22      </sequence>
23      <while condition="si">
24        <invoke name="Cancelar Actividad" partnerLink="local"
25  portType="localPT" operation="Cancelar Actividad" inputVariable="Cancelar
26  Actividad_data_in" outputVariable="Cancelar Actividad_data_out" />
27      </while>
28    </sequence>
29  </process>
```

Figura 32: WS-BPEL inicial de la reserva de actividades

3.3. Generación del proceso WS-BPEL completo

El WS-BPEL generado por BABEL a partir del modelo BPMN es incompleto. Se necesita información adicional del usuario para poder completarlo y que este pueda ser desplegado en el servidor y consumido por otras aplicaciones.

Para ello, se ha implementado una herramienta gráfica basada en formularios que permite al usuario introducir la información faltante de una forma sencilla e intuitiva. Además se han generado una serie de transformaciones para: 1) adaptar el resultado de BABEL a la herramienta gráfica implementada; 2) completar el WS-BPEL generado por BABEL; y 3) generar los artefactos esenciales asociados al proceso.

Este apartado esta formado por seis secciones: 1) la primera de ellas explica el metamodelo implementado para la propuesta, especificando las decisiones tomadas en su implementación y los diferentes elementos que lo componen; 2) en segundo lugar, se explica la transformación automática del WS-BPEL inicial al modelo intermedio; 3) en tercer lugar, se explica la herramienta para completar la información del modelo intermedio con la información del usuario; 4) en cuarto lugar, se describe como se completa el WS-BPEL inicial con la información del modelo intermedio; 5) en quinto lugar, se explica la generación de WSDL del proceso a partir del modelo intermedio; y 6) por último, se describe la generación del fichero deploy.xml a partir del modelo intermedio. El fichero deploy.xml será necesario para el correcto despliegue de la composición de servicios en el servidor.

3.3.1. Metamodelo intermedio

Para permitir al usuario completar la información del WS-BPEL generado por BABEL, se ha implementado un metamodelo intermedio cuya función es almacenar la información del WS-BPEL inicial de forma que sea fácilmente modificable por el usuario.

El metamodelo intermedio se puede ver en la Figura 33. Éste ha sido definido de manera que no almacena toda la información de un proceso WS-BPEL, solo almacena la información necesaria para que el usuario pueda completar el WS-BPEL inicial usando la herramienta gráfica. El resto de la información permanece en el WS-BPEL inicial. Posteriormente, el WS-BPEL inicial es completado con la información contenida en el modelo intermedio ya refinado por el usuario.

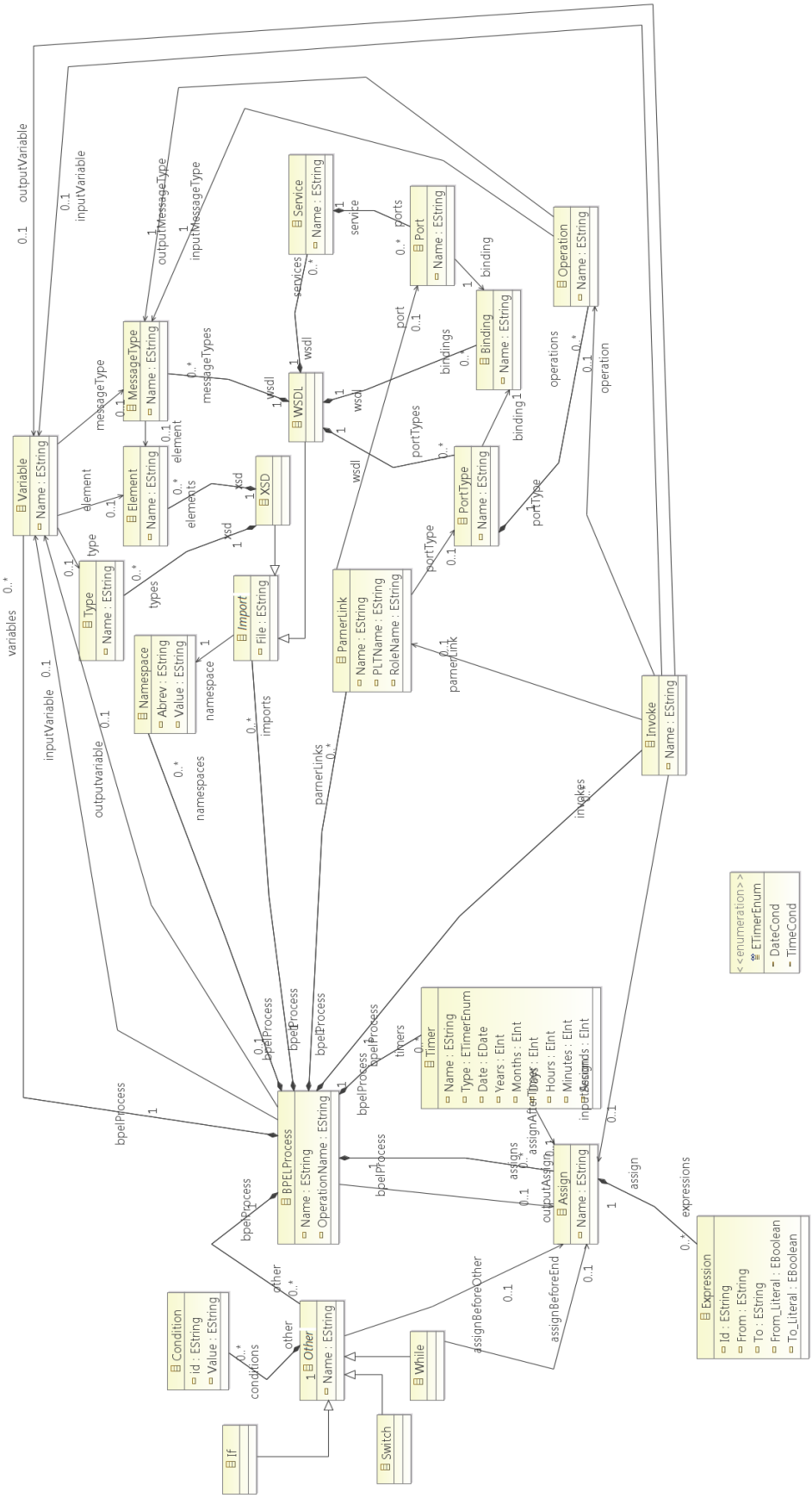

Figura 33: Metamodelo intermedio

En la Tabla 1 se explican las diferentes metaclases que conforman el metamodelo y las relaciones entre ellas.

BPELProcess
Ésta es la metaclase principal. Representa al elemento proceso de WS-BPEL y contiene al resto de metaclases.
Atributos
Nombre (Name): Permite especificar el nombre del proceso principal. **Nombre de operación (OperationName):** Permite especificar el nombre que se le asigna a la operación que realiza el proceso. Este nombre se define en el WSDL del proceso WS-BPEL.
Relaciones*
Variable de entrada (InputVariable): Especifica la variable de entrada del proceso. **Variable de salida (OutputVariable):** Especifica la variable de salida del proceso. **Assign de salida (OutputAssign):** Permite manipular los datos y dar el valor deseado a la variable de salida del proceso.
**El resto de relaciones tienen la única función de englobar al resto de metaclases.*
Other
Ésta es la metaclase padre de las metaclases *While* y *Switch*, permite agrupar los atributos y relaciones comunes entre ambos hijos.
Atributos
Nombre: Especifica el nombre del elemento.
Relaciones
Condiciones (Conditions): Permite especificar la expresión XPATH que define el comportamiento de este tipo de elementos. **Assign antes del elemento (AssignBeforeElement):** Permite especificar un *Assign* antes del elemento para manipular los datos que posteriormente son utilizados por el elemento.
While
Metaclase encargada de representar al elemento *While* de WS-BPEL.
Relaciones
Assign antes de terminar (AssignBeforeEnd): Permite especificar un *Assign*, el cual es ejecutado una vez por cada iteración.
Switch
Metaclase encargada de representar el elemento *Switch* de WS-BPEL
Condition
Metaclase utilizada para especificar el comportamiento de las condiciones que contienen los elementos *While* y *Switch* en WS-BPEL.
Atributos
Id: Permite identificar una condición de forma inequívoca. **Valor (Value):** Expresión XPATH que especifica su comportamiento.
Assign
Metaclase que representa al elemento *Assign* de WS-BPEL
Atributos
Nombre (Name): Permite especificar el nombre del *Assign*.

Relaciones

Expresiones (Expressions): Permite especificar las expresiones asociadas al elemento *Assign*, las cuales permitirán manipular los datos y dar valor a las variables del proceso.

Expression

Metaclase encargada de especificar los diversos comportamientos que puede tener un *Assign.*

Atributos

Desde (From): Permite especificar y manipular los datos que serán copiados en la variable especificada en el atributo *To*.

Hacia (To): Permite especificar la variable a la cual se le copiaran los datos especificados en el *From*.

Desde literal (From_Literal): Atributo booleano que especifica si el *From* es literal, es decir, si éste debe tomarse como texto plano o como una expresión.

Hacia literal (To_Literal): Atributo booleano que especifica si el *To* es literal, es decir, si éste debe tomarse como texto plano o como una expresión.

Timer

Metaclase que representa al elemento *Wait* de WS-BPEL.

Atributos

Nombre (Name): Permite especificar el nombre del elemento.

Fecha (Date): Permite especificar una fecha hasta la cual el elemento tiene que esperar.

Año (Year): Permite especificar el número de años que el elemento tiene que esperar.

Mes (Month): Permite especificar el número de meses que el elemento tiene que esperar

Días (Day): Permite especificar el número de días que el elemento tiene que esperar.

Horas (Hours): Permite especificar el número de horas que el elemento tiene que esperar o la hora hasta la que tiene que esperar.

Minutos (Minutes): Permite especificar el número de minutos que el elemento tiene que esperar o el minuto hasta el que tiene que esperar.

Segundos (Seconds): Permite especificar el número de segundos que el elemento tiene que esperar o el segundo hasta el que tiene que esperar.

Tipo (Type): Este atributo es un elemento de tipo enumeración, el cual puede tener dos valores: 1) *DateCond*: El elemento espera hasta un fecha concreta; y 2) *TimeCond*: El elemento espera una cantidad de tiempo determinada.

Variable

Metaclase que representa al elemento *Variable* de WS-BPEL.

Atributos

Nombre (Name): Permite especificar el nombre de la variable.

Relaciones*

Tipo (Type): Permite especificar el tipo de la variable.

Elemento (Element): Permite especificar el tipo de elemento que contiene una variable.

Tipo de mensaje (MessageType): Permite especificar el tipo de mensaje de la variable.

**Una variable solo puede tener un tipo, un elemento o un tipo de mensaje.*

Type

Metaclase que representa un tipo de datos simple que define una variable.

Atributos

Nombre (Name): Permite especificar el nombre del tipo.

Element
Metaclase que representa el elemento que define una variable.
Atributos
Nombre (Name): Permite especificar el nombre del elemento.
MessageType
Metaclase que presenta un tipo de mensaje que define una variable
Atributos
Nombre (Name): Permite especificar el nombre del tipo de mensaje.
Import
Metaclase padre que engloba a las metaclases *XSD* y *WSDL*.
Atributos
Archivo (File): Especifica la ruta donde se encuentra el archivo importado.
Relaciones
Espacio de nombre (Namespace): Especifica el *Namespace* asociado al archivo importado.
XSD
Metaclase que representa los *XSDs* importados en el proceso.
Relaciones
Tipos (Types): Permite especificar los elementos de tipo *Type* que son especificados en el *XSD*. **Elementos (Elements):** Permite especificar los elementos de tipo *Element* que son especificados en el *XSD*.
WSDL
Metaclase que representa los *WSDLs* importados en el proceso.
Relaciones
Tipo de mensajes (MessageTypes): Permite especificar los tipos de mensaje definidos en el *WSDL*. **Tipos de puerto (PortTypes):** Permite especificar los *PortTypes* definidos en el *WSDL*. **Ligaduras (Bindings):** Permite especificar los *Bindings* definidos en el *WSDL*. **Servicios (Services):** Permite especificar los *Services* definidos en el *WSDL*.
Namespace
Metaclase que representa al tipo de elemento *Namespace* dentro del proceso WS-BPEL.
Atributos
Abreviatura (Abrev): Permite especificar la abreviatura que permite hacer referencia al *Namespace*. **Valor (Value):** Permite especificar el *Namespace* completo.
PortType
Metaclase que representa los tipos de puerto de un *WSDL.*
Atributos
Nombre (Name): Permite especificar el nombre del tipo de puerto.
Relaciones
Operaciones (Operations): Permite especificar las operaciones contenidas en el *PortType*.
Binding
Metaclase que representa las ligaduras definidas en un *WSDL.*

Atributos
Nombre (Name): Permite especificar el nombre de la ligadura.
Port
Metaclase que representa los puertos definidos en un *WSDL.*
Atributos
Nombre (Name): Permite especificar el nombre del puerto.
Relaciones
Ligadura (Binding): Permite especificar la ligadura asociada a un puerto.
Service
Metaclase que representa los servicios definidos en un *WSDL.*
Atributos
Nombre (Name): Permite especificar el nombre del servicio.
Relaciones
Puertos (Ports): Permite especificar los puertos asociados a un servicio.
Operation
Metaclase que representa las operaciones definidas en un *WSDL*.
Atributos
Nombre (Name): Permite especificar el nombre de la operación.
Relaciones
Tipo de mensaje de entrada (InputMessageType): Permite especificar el tipo de mensaje de entrada de la operación. **Tipo de mensaje de salida (OutputMessageType):** Permite especificar el tipo de mensaje de salida de la operación.
Invoke
Metaclase que representa el elemento *Invoke* de WS-BPEL.
Atributos
Nombre (Name): Permite especificar el nombre del *Invoke*.
Relaciones
ParnerLink: Permite especificar el *ParnerLink* asociado a un *Invoke*. **Operación (Operaction):** Permite especificar la operación que realiza un *Invoke*. **Variable de entrada (IputVariable):** Permite especificar la variable de entrada de un *Invoke*. **Variable de salida (OutputVariable):** Permite especificar la variable de salida de un *Invoke*.
ParnerLink
Metaclase que representa el *ParnerLink* de un WS-BPEL.
Atributos
Nombre (Name): Permite especificar el nombre del *ParnerLink*. **Nombre del PLT (PLTName):** Permite especificar el nombre del tipo de *ParnerLink*, **Nombre del role (RoleName):** Permite especificar el nombre del *role*.
Relaciones
Tipo de puerto (PortType): Permite especificar el tipo de puerto asociado al *ParnerLink* **Puerto (Port):** Permite especificar el tipo de puerto asociado al *ParnerLink*.

Tabla 1: Especificación de las metaclases de metamodelo intermedio

3.3.2. Transformación del WS-BPEL inicial en un modelo intermedio

El objetivo de la transformación automática del WS-BPEL inicial en un modelo intermedio es transformar el resultado de la transformación BABEL explicada en el apartado 3.2, en un modelo entendible por la herramienta gráfica basada en FMF, la cual es utilizada por el usuario, para la introducción de la información restante. El modelo resultante de esta transformación es una instancia del metamodelo explicado en el apartado 3.3.1, sobre el cual trabaja la herramienta.

Para la implementación de esta transformación, se ha utilizado como lenguaje Java, utilizando las librerías de JDOM y el lenguaje XPATH los cuales facilitan en gran medida la manipulación de documentos XML (tales como, WS-BPEL, WSDL, XMI).

Ejemplo

Continuando con el ejemplo mostrado en el apartado 3.1, se ejecuta la transformación del WS-BPEL inicial mostrado en la Figura 32 en el modelo intermedio.

Esta transformación recorre el WS-BPEL inicial convirtiendo cada uno de sus elementos en su equivalente en el metamodelo. Además, crea varios elementos por defecto que facilitan al usuario final la tarea de completar el proceso. Estos elementos son: 1) las variables de salida y de entrada del proceso principal; 2) el WSDL del proceso; 3) un XSD con todos los elementos habituales utilizados en un proceso WS-BPEL; y 4) los *Namespace* que hacen referencia al WSDL y al XSD generados.

En la Figura 34 se puede ver el resultado de la transformación. El resultado se muestra mediante la representación visual que proporciona el proyecto EMF por defecto para facilitar la lectura y edición de los modelos.

En el modelo resultante de la transformación se pueden diferenciar cinco grupos de elementos: 1) en primer lugar, los elementos de tipo *Invoke* obtenidos mediante la transformación de cada aparición del elemento *<invoke>* en el WS-BPEL inicial; 2) en segundo lugar, se hallan dos elementos de tipo *Import*, concretamente uno de tipo *WSDL* y otro de tipo *XSD*. Como se ha mencionado, éstos son generados automáticamente para facilitar la introducción de información por parte del usuario; 3) en tercer lugar, dos elementos de tipo *Other*, concretamente de tipo *While*, que son el resultado de la transformación de cada aparición del elemento <*while*> en el WS-BPEL inicial; 4) en cuarto lugar, se encuentran la dos variables utilizadas para indicar la entrada y salida del proceso, respectivamente. Al igual que los elementos *WSDL* y *XSD,* éstas son generadas automáticamente; 5) por último, se encuentran los dos elementos de tipo *Namespace* generados automáticamente, permitiendo referenciar a los elementos *WSDL* y *XSD* mencionados anteriormente.

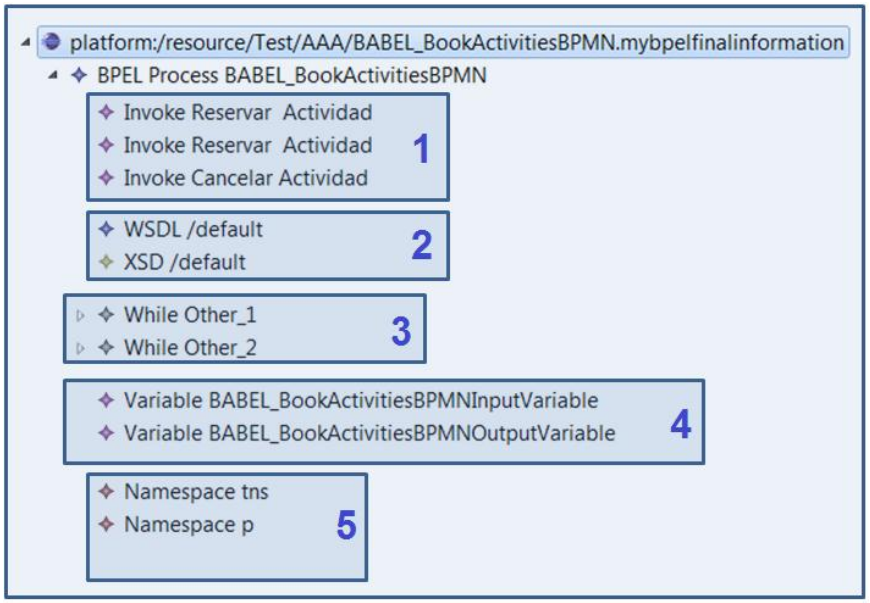

Figura 34: Diagrama EMF del modelo intermedio en la reserva de actividades

3.3.3. Edición del modelo intermedio mediante una interfaz

En esta fase, el usuario introduce la información necesaria para completar el WS-BPEL inicial con ayuda de la herramienta gráfica implementada basada FMF. Se ha decido utilizar una herramienta gráfica basada en formularios, principalmente porque ésta ofrece un mecanismo sencillo para añadir información. Esto hace que la propuesta pueda ser mas fácilmente introducida dentro de un entorno industrial, y que la curva de aprendizaje del usuario sea mas baja. Además, gracias al potente framework FMF, se han desarrollado los formularios que forman la herramienta, de una manera sencilla, rápida y muy eficiente.

La información que el usuario debe introducir para completar el proceso es: 1) información referente a los servicios Web tales como *Partner links*, *Port Type*, variables de entrada, variables de salida, WSDLs y XSDs; 2) información para la manipulación de los datos, es decir, la especificación de los *Assigns;* y 3) la especificación de la expresiones que definen el comportamiento de los elementos de tipo *While*, *Switch* y *Timer*.

El patrón que se utiliza para la gran mayoría de pestaña a excepción del que se utiliza para representar la información del proceso principal, divide la pantalla en dos secciones: 1) la “sección A” muestra una lista con todos los elementos de un tipo concreto y en los casos necesarios incluye botones para la creación y eliminación de elementos de la lista; y 2) la “sección C” muestra los atributos asociados al elemento seleccionado en la “sección lista”, cuyo valor puede ser modificado por el usuario.

Además, en la parte inferior de ésta se pueden ver nueve pestañas, las cuales representan los elementos principales del proceso y permiten el acceso a sus respectivos formularios. Por ejemplo, en la pestaña *Process* se encuentra el formulario asociado a un

elemento de tipo *Process*, mostrado en la Figura 36. A diferencia de lo que se ha comentado hasta el momento, éste es el único formulario que sigue una estructura diferente. Esto se debe a que solo puede existir un elemento de tipo *Process*, por lo que la "sección C" con la lista es suprimida y solo se muestra la sección de detalle con sus respectivos atributos y la "sección B" que permite navegar entre las diferentes pestañas.

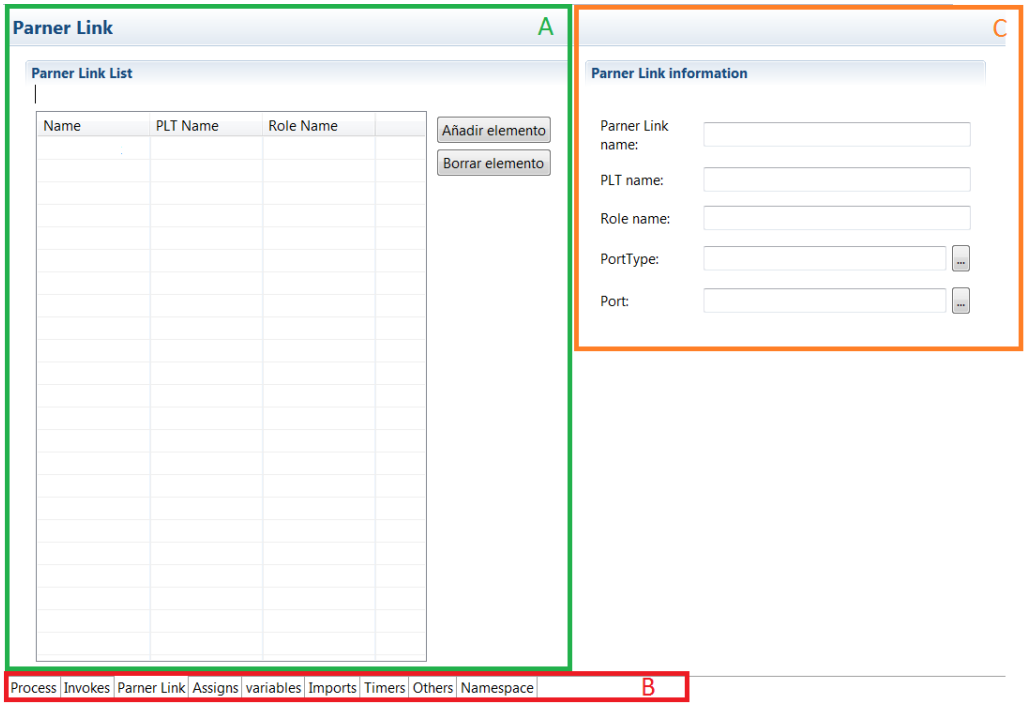

Figura 35: Formulario para manejar los ParnerLinks

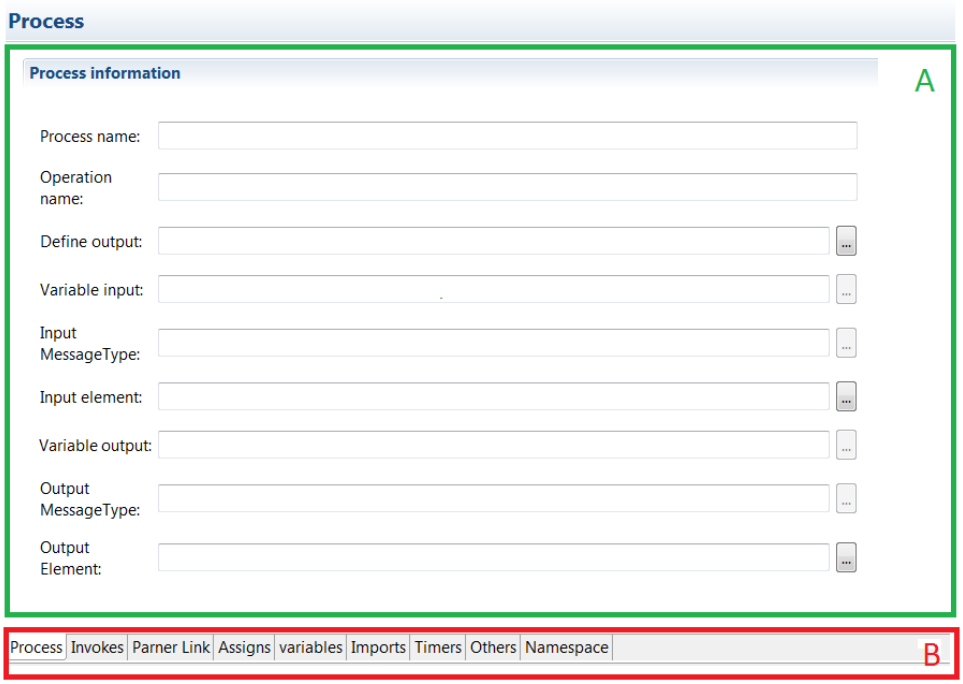

Figura 36: Formulario para administrar el Process

Ejemplo

Para explicar cómo el usuario utiliza la herramienta para completar el proceso WS-BPEL que ha generado BABEL, se describe el proceso a seguir para introducir la información de algunos de los elementos principales del WS-BPEL. Por ejemplo, para completar la información del *Invoke* "Reserva Actividad", el usuario debe seguir tres pasos:

1) Importar el WSDL y los XSDs (en caso de ser necesario) que definen el servicio Web invocado por el *Invoke*. Para ello, el usuario tiene que ir a la pestaña "Imports", la cual se muestra en la Figura 37, y añadir el WSDL del servicio Web.

2) Una vez el WSDL ha sido importado, el siguiente paso es crear el *ParnerLink* que posteriormente será asociado al *Invoke*. Para ello, el usuario abre la pestaña "Partner Link" mostrada en la Figura 38. Luego, se elige crear un nuevo *PartnerLink* y se asignan los correspondientes valores a sus atributos.

3) Por último, se asigna el *PartnerLink* creado al *Invoke* con el fin de especificar el servicio y la operación invocada por éste. Además, se le asigna una variable de entrada y salida al *Invoke*. Para ello, hay dos opciones: a) definir las variables y luego asignar los correspondientes valores a los atributos del *Invoke*; o b) utilizar la funcionalidad implementada por la herramienta, la cual en función de la operación seleccionada para el *Invoke,* genera automáticamente las variables de entrada y de salida. En el ejemplo se utiliza la segunda opción. Por consiguiente, el usuario debe ir a la pestaña "Invokes" y rellenar los atributos, como se muestra en la Figura 39.

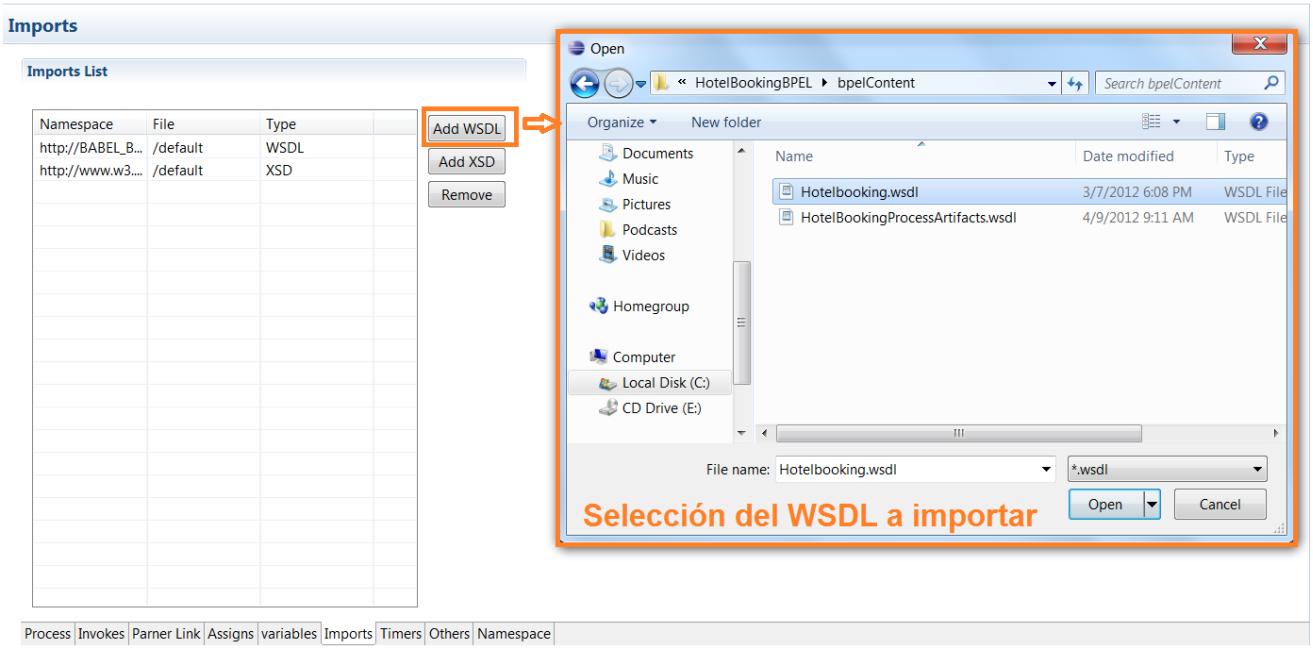

Figura 37: Formulario para administrar los Imports

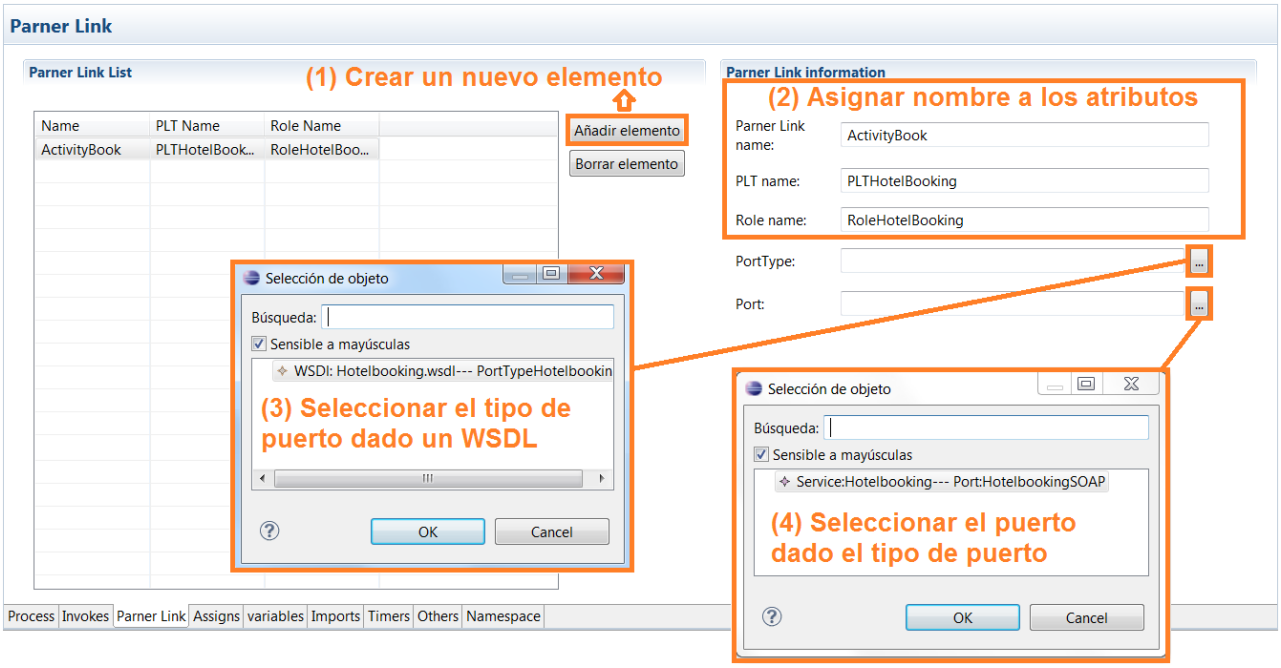

Figura 38: Formulario para administrar los ParnerLinks remarcando los pasos de creación de un nuevo ParnerLink

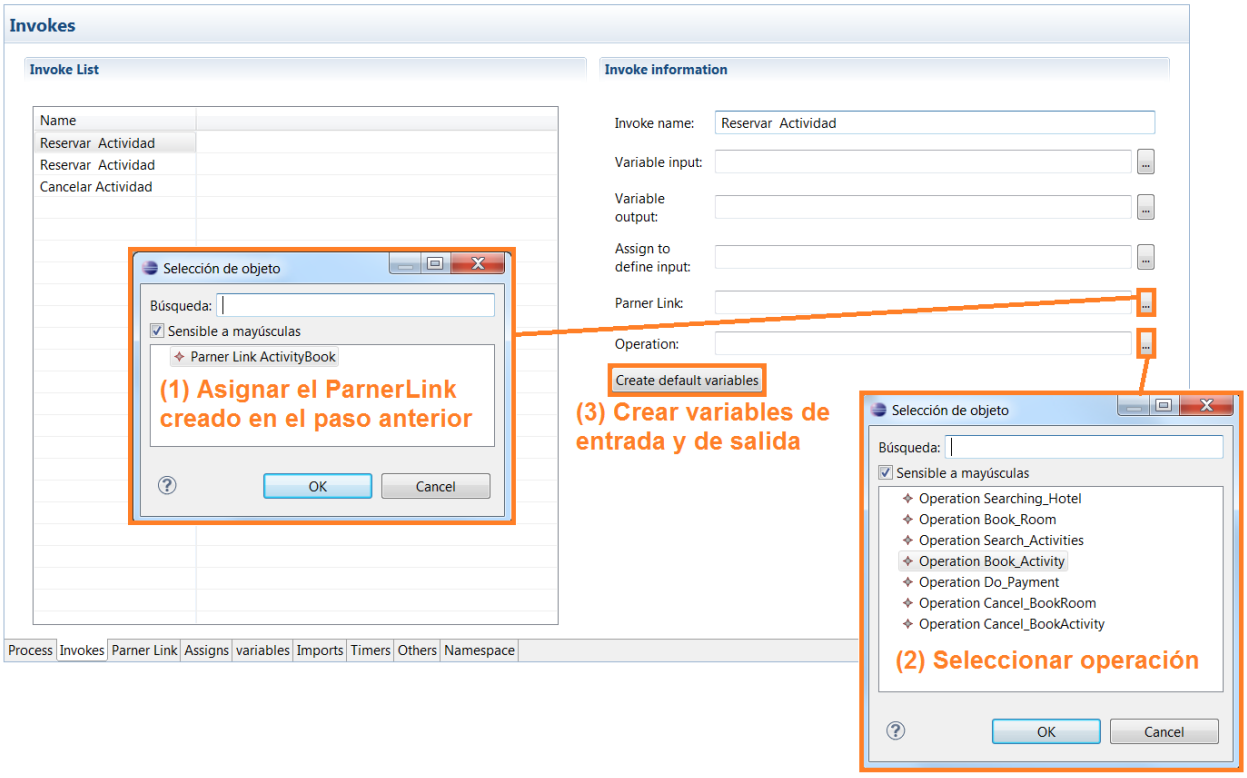

Figura 39: Formulario para administrar los Invokes

Una vez definidos todos los *Invokes*, se debe continuar con los *Assigns*, las condiciones de los *Whiles* y por último especificar la entrada y la salida del proceso principal. El resultado después de haber realizado todas las operaciones necesarias se puede ver en la Figura 40.

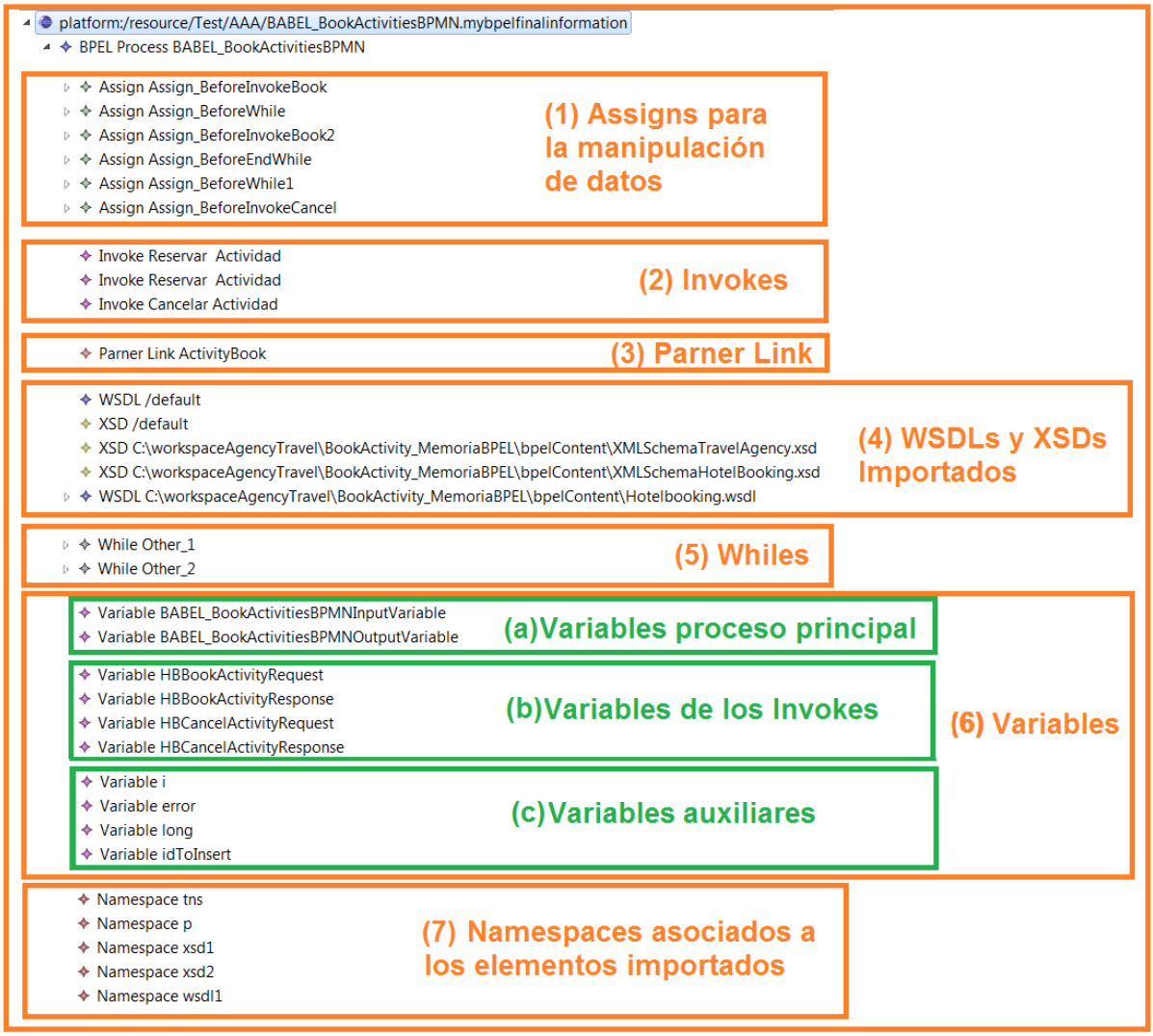

Figura 40: Modelo intermedio completado para la reserva de actividades

En la Figura 40 se pueden diferenciar siete grupos de elementos:

1) Los seis *Assigns* definidos en el proceso. Éstos permiten manipular los datos y dar valor a las variables. Por ejemplo, el primer *Assign* realiza el mapeo entre la variable de entrada al proceso y la variable de entrada del *Invoke* encargado de hacer la reserva de una actividad.

2) Los *Invokes* han sido completados con la información de sus *ParnerLinks*, las operaciones que realizan y sus variables. Éstas últimas se pueden ver en la sección "Variables de los *Invokes*".

3) El único *ParnerLink* del proceso permite la invocación de los servicios.

4) Los diferentes elementos importados en el proceso: a) Elementos por defecto. Su misión es facilitar al usuario la definición del proceso; b) El WSDL que representa al servicio Web invocado durante la ejecución del proceso; c) El XSD donde se definen los tipos del servicio Web; y d) El XSD donde se han definido elementos y tipos

utilizados para la definición del proceso de reserva de actividades. En este caso, el XSD ha sido utilizado para la definición de tipos de elemento utilizado como entrada y salida del proceso principal.

5) Los *Whiles* ya completados con la definición de su comportamiento.

6) Las variables utilizadas a lo largo del proceso que se pueden separar en tres grupos: a) Variables del proceso principal: representan la entrada y la salida del proceso; b) Variables de los *Invokes*: representan la entrada y la salida de cada uno de los *Invokes*; y c) Variables auxiliares: permiten almacenar información temporalmente y por tanto, especificar comportamientos más complejos. Por ejemplo, la variable "i" que funciona como contador dentro de un *While* y la variable "long" que almacena el número de elementos de una lista, permiten recorrer una lista de elementos.

7) Los *Namespaces* permiten hacer referencia dentro del proceso, por ejemplo dentro de un *Assign*, a los elementos contenidos dentro de los diferentes elementos importados.

3.3.4. Transformación del modelo intermedio al WS-BPEL final

Una vez el usuario ha completado la información contenida en el modelo intermedio, mediante la utilización de la herramienta gráfica, se ejecuta la transformación automática al WS-BPEL final. Ésta es la encargada de completar el WS-BPEL inicial, obtenido mediante la transformación BABEL, con la información introducida por el usuario en el modelo intermedio. En el modelo BPMN mostrado en la Figura 41 se pueden ver las entradas y la salida de esta transformación.

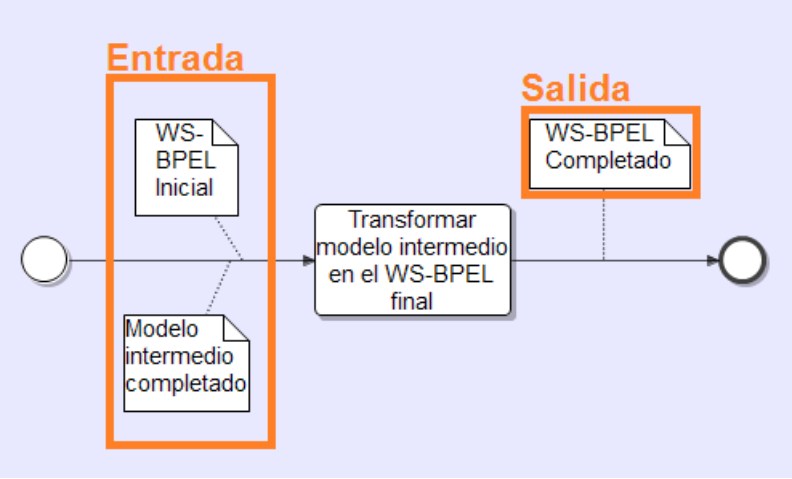

Figura 41: Diagrama BPMN de la transformación del modelo intermedio al WS-BPEL Final

Concretamente, para la implementación de esta transformación se ha utilizado Java como lenguaje y la librería JDOM y el lenguaje XPATH para facilitar las búsquedas y las manipulaciones en los documentos XMLs.

Básicamente, los pasos seguidos por la transformación para completar el WS-BPEL inicial son los siguientes: 1) la herramienta busca un grupo de elementos en el modelo intermedio, por ejemplo los *Invokes*; 2) dado un grupo de elementos de un tipo, busca el equivalente en el WS-BPEL inicial; y 3) cada uno de los elementos del WS-BPEL inicial son completados con la información de su equivalente en el modelo intermedio.

Ejemplo

Continuando con el ejemplo de la reserva de actividades del apartado 3.3.3, el código resultante de la ejecución de la transformación a código WS-BPEL completo se encuentra en el DVD adjunto. Para facilitar el entendimiento de esta transformación, sólo se muestran secciones del código final.

Por ejemplo, para completar el *Invoke* cuyo nombre es "Reservar actividad" se tiene como entrada: 1) el documento WS-BPEL inicial, cuya sección de código referente al *Invoke* se muestra en la Figura 42; y 2) el modelo intermedio completado. La información de este último es mostrada en la Figura 43 a través de la herramienta gráfica utilizada para su edición.

```
<invoke name="Reservar  Actividad"
partnerLink="local" portType="localPT"
operation="Reservar  Actividad"
inputVariable="Reservar  Actividad_data_in"
outputVariable="Reservar  Actividad_data_out"
/>
```

Figura 42: Sección del *Invoke* Reservar Actividad en el WS-BPEL inicial

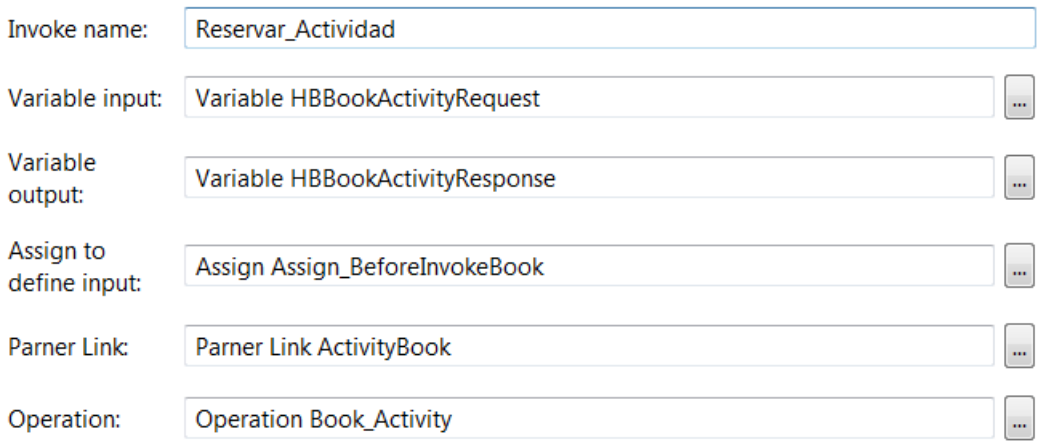

Figura 43: Formulario de edición de Invokes con la información del *Invoke* Reservar Actividad

El resultado de aplicar la transformación se puede ver en la Figura 44. Básicamente, se ha completado cada uno de los atributos del *Invoke* con la información contenida en el modelo intermedio.

```
<bpel:invoke name="Reservar_Actividad"
partnerLink="ActivityBook"
portType="wsdl1:Hotelbooking"
operation="Book_Activity"
inputVariable="HBBookActivityRequest"
outputVariable="HBBookActivityResponse"
/>
```

Figura 44: Sección del *Invoke* Reservar Actividad en el WS-BPEL completo

Además, para que el Invoke quede operativo, es necesario que se creen las *Variables* y el *ParnerLink* a los que hace referencia. Además, se debe añadir un elemento de tipo *Import* que especifica el WSDL donde se define el *ParnerLink Type*.

El procedimiento seguido por la transformación es similar para todos los elementos: 1) se recolecta la información del modelo intermedio; y 2) se crea o se completa el elemento en el WS-BPEL inicial, convirtiendo a éste en el WS-BPEL final cuando todos los elementos han sido completados y creados.

3.3.5. Transformación del modelo intermedio al WSDL del proceso

Esta transformación, al igual que la transformación para completar el WS-BPEL inicial, es ejecutada después de que el usuario completa la información con la herramienta gráfica. Esta transformación utiliza el lenguaje Java, junto con la librería JDOM y el lenguaje XPATH para recolectar la información del modelo intermedio y generar el documento WSDL del documento final WS-BPEL. La Figura 45 muestra la representación gráfica BPMN de esta transformación.

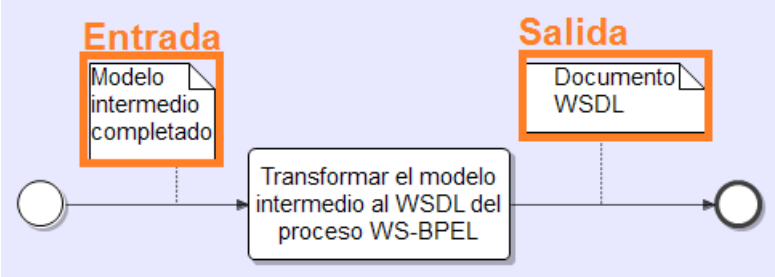

Figura 45: Diagrama BPMN de la transformación del modelo intermedio al WSDL del proceso WS-BPEL

La transformación sigue una serie de pasos: 1) la herramienta genera la estructura de un documento WSDL básico, común a todos los documentos de este tipo; 2) la herramienta añade todos lo *Imports* que posteriormente serán utilizados en la definición del documento y los *Namespaces* que permiten hacer referencia a los *Imports*; 3) se definen los mensajes

que describen la operación implementada por el proceso WS-BPEL; 4) la herramienta añade la definición del *PortType* y la operación asociada al *PortType*, especificando los mensajes de entrada y de salida de la operación; 5) se definen los *ParnerLink Types* utilizados durante el proceso, incluido el del propio proceso; y 6) por último, se genera el *Binding* y el *Service* a partir de los elementos introducidos anteriormente.

Ejemplo

Continuando con el ejemplo de la reserva de actividades, dado el modelo intermedio completado con la información del usuario, la herramienta genera el documento WSDL referente al proceso WS-BPEL. El código resultante de la transformación del modelo intermedio se puede ver en la Figura 46.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
3  xmlns:p="http://www.w3.org/2001/XMLSchema" xmlns:plnk="http://docs.oasis-
4  open.org/wsbpel/2.0/plnktype"
5  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
6  xmlns:tns="http://BABEL_BookActivitiesBPMN"
7  xmlns:wsdl1="http://www.example.org/Hotelbooking/"
8  xmlns:xsd1="http://www.example.org/XMLSchemaTravelAgency"
9  xmlns:xsd2="http://www.example.org/HotelXMLSchema"
10 name="BABEL_BookActivitiesBPMN"
11 targetNamespace="http://BABEL_BookActivitiesBPMN">
12
13   <wsdl:import location="Hotelbooking.wsdl"
14 namespace="http://www.example.org/Hotelbooking/" />
15
16   <wsdl:types>
17     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
18       <xsd:import schemaLocation="XMLSchemaTravelAgency.xsd"
19 namespace="http://www.example.org/XMLSchemaTravelAgency" />
20     </xsd:schema>
21   </wsdl:types>
22
23   <wsdl:message name="BABEL_BookActivitiesBPMNOutputMessage">
24     <wsdl:part element="xsd1:ResponseBookAct" name="parameters" />
25   </wsdl:message>
26   <wsdl:message name="BABEL_BookActivitiesBPMNInputMessage">
27     <wsdl:part element="xsd1:RequestBookActivity" name="parameters" />
28   </wsdl:message>
29
30   <wsdl:portType name="BABEL_BookActivitiesBPMN">
31     <wsdl:operation name="BookActivity">
32       <wsdl:input message="tns:BABEL_BookActivitiesBPMNInputMessage" />
33       <wsdl:output message="tns:BABEL_BookActivitiesBPMNOutputMessage" />
34     </wsdl:operation>
35   </wsdl:portType>
36
37   <plnk:partnerLinkType name="PLTHotelBooking">
38     <plnk:role name="RoleHotelBooking" portType="wsdl1:Hotelbooking" />
39   </plnk:partnerLinkType>
40   <plnk:partnerLinkType name="BABEL_BookActivitiesBPMN">
41     <plnk:role name="BABEL_BookActivitiesBPMNProvider"
42 portType="tns:BABEL_BookActivitiesBPMN" />
```

```
43    </plnk:partnerLinkType>
44
45    <wsdl:binding name="BABEL_BookActivitiesBPMNBinding"
46  type="tns:BABEL_BookActivitiesBPMN">
47      <soap:binding style="document"
48  transport="http://schemas.xmlsoap.org/soap/http" />
49      <wsdl:operation name="BookActivity">
50        <soap:operation
51  soapAction="http://BABEL_BookActivitiesBPMN/BookActivity" />
52        <wsdl:input>
53          <soap:body use="literal" />
54        </wsdl:input>
55        <wsdl:output><soap:body use="literal" /></wsdl:output>
56      </wsdl:operation>
57    </wsdl:binding>
58    <wsdl:service name="BABEL_BookActivitiesBPMN">
59      <wsdl:port name="BABEL_BookActivitiesBPMNPort"
60  binding="tns:BABEL_BookActivitiesBPMNBinding">
61        <soap:address
62  location="http://localhost:8081/ode/processes/BABEL_BookActivitiesBPMN" />
63      </wsdl:port> </wsdl:service> </wsdl:definitions>
```

Figura 46: Documento WSDL del proceso WS-BPEL para la reserva de actividades

A continuación se describen algunos de los elementos más relevantes del modelo intermedio, mostrando como éstos han influido en la generación del documento WSDL de la Figura 46 .

Por ejemplo, uno de los elementos fundamentales para la generación del WSDL, es el *BPELProcess*. En este elemento se definen los mensajes, el tipo de elemento que define cada uno de los mensajes y el nombre de la operación. En la Figura 47 se puede ver la información en este ejemplo concreto.

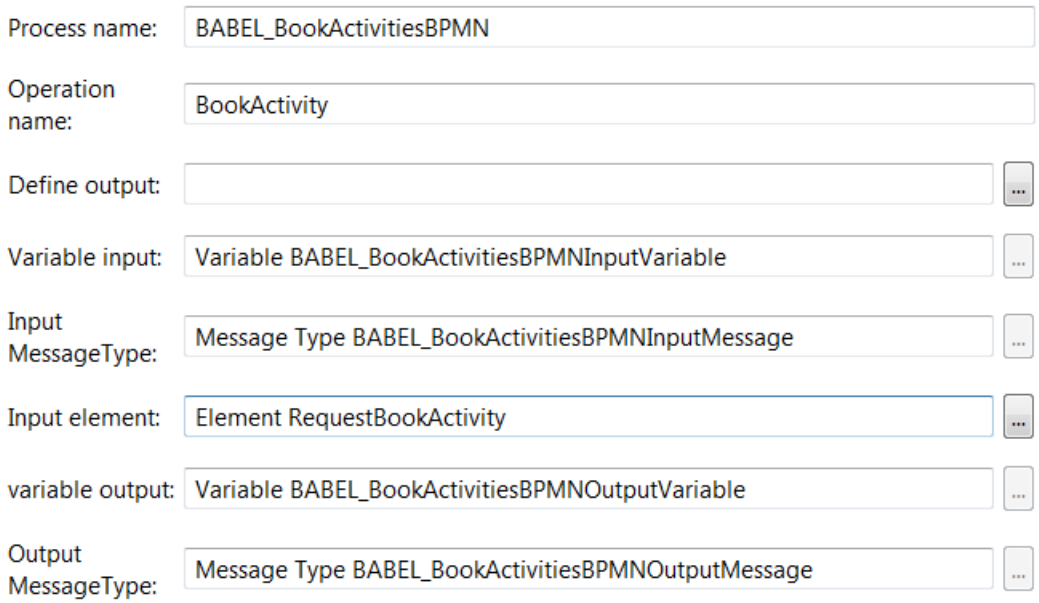

Figura 47: Formulario de edición del proceso con la información del proceso de reserva de actividades

A partir de esta información se generan: 1) los mensajes y la operación que utiliza estos mensajes. Esto se puede ver en la Figura 46, entre la líneas 25-30 y 32-37 respectivamente; y 2) la importación del o los XSDs que definen el tipo de elemento utilizado por los mensajes. Concretamente, en este caso es un único XSD que se puede ver entre las líneas 16-21 de la Figura 46.

Además de la información proporcionada por el elemento *BPELProcess*, se genera un *ParnerLink Type* por cada elemento *ParnerLink* del modelo intermedio y uno generado automáticamente para representar al propio proceso. En este caso, existen dos definiciones, una por el *ParnerLink* que permite la invocación de los servicios Web de reserva y cancelación de actividades y otro por el proceso de reserva de actividades. Ambos pueden ser vistos entre las líneas 37-43 de la Figura 46.

Finalmente, el resto de elementos tales como el *Binding* y el *Service* son generados automáticamente utilizando como única información la que contiene el propio WSDL.

3.3.6. Transformación del modelo intermedio al Deploy.xml

Al igual que las dos últimas transformaciones, esta transformación es ejecutada una vez que el usuario ha terminado de introducir la información al modelo intermedio. Esta transformación recolecta la información del modelo intermedio generando un documento XML llamado "deploy" el cual permite desplegar el proceso WS-BPEL en el servidor. En la Figura 48 se puede ver el diagrama BPMN que representa esta transformación.

Figura 48: Diagrama BPMN de la transformación del modelo intermedio al deploy.xml

Básicamente se genera una sección de código en el fichero deploy.xml por cada uno de los *ParnerLinks* definidos en el modelo intermedio. En esta sección de código se especifica el *Port* concreto que utiliza el *ParnerLink* dentro del fichero WS-BPEL. Esta información es necesaria ya que para un mismo servicio Web pueden existir diferentes puertos, por lo que es necesario especificar el puerto usado para el proceso concreto.

Ejemplo

En el proceso definido para la implementación de la reserva de actividades existen únicamente dos *ParnerLinks:* 1) el definido para el proceso principal; y 2) el utilizado para hacer las llamadas a los servicios Web de cancelar y reservar actividades.

En el primer caso, el puerto es generado automáticamente por la transformación encargada de la generación del WSDL. Como la generación del WSDL no tiene por qué ser ejecutada antes de esta transformación, se ha decidido que el nombre del puerto generado siempre siga la siguiente estructura: "Nombre del proceso + Puerto". Con ello se consigue que independientemente del orden de las transformaciones, siempre se tenga el nombre del puerto del proceso principal, permitiendo completar la generación del deploy.xml. En el segundo caso, es necesario recolectar la información del modelo intermedio. Como se puede ver en la Figura 49, esta información es introducida por el usuario cuando se crea un nuevo elemento de tipo *ParnerLink.*

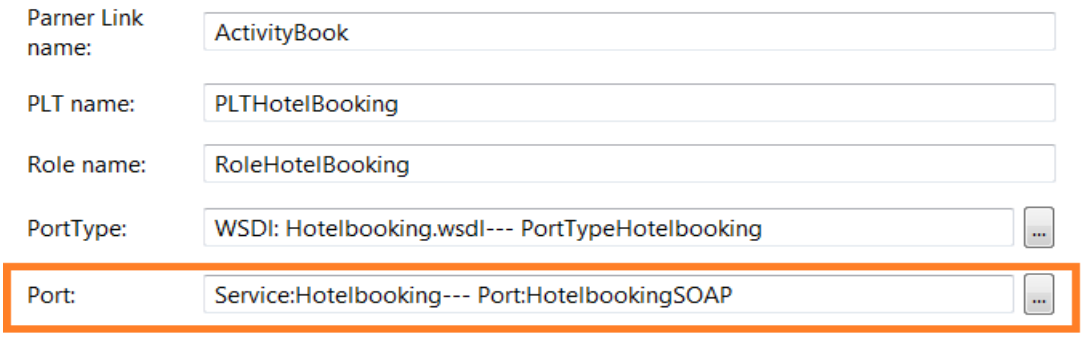

Figura 49: Formulario de edición de un *ParnerLink* con la información de *ParnerLink* Reserva de Actividad

Dada la información explicada se genera el deploy.xml, cuyo código se puede ver en la Figura 50.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <deploy:deploy xmlns:deploy="http://www.apache.org/ode/schemas/dd/2007/03"
3  xmlns:tns="http://BABEL_BookActivitiesBPMN"
4  xmlns:wsdl1="http://www.example.org/Hotelbooking/">
5    <deploy:process name="tns:BABEL_BookActivitiesBPMN">
6      <deploy:active>true</deploy:active>
7      <deploy:retired>false</deploy:retired>
8      <deploy:process-events generate="all" />
9
10    <deploy:provide partnerLink="client">
11       <deploy:service
12          name="tns:BABEL_BookActivitiesBPMN"
13          port="BABEL_BookActivitiesBPMNPort" />
14     </deploy:provide>
15
16     <deploy:invoke partnerLink="ActivityBook">
17       <deploy:service
18          name="wsdl1:Hotelbooking"
```

```
19              port="HotelbookingSOAP" />
20          </deploy:invoke>
21
22        </deploy:process>
23    </deploy:deploy>
```

Figura 50: Documento deploy.xml asociado al proceso de la reserva de actividades

3.4. Despliegue en caliente

El despliegue en caliente es una de las características más importantes de la herramienta, ya que es fundamental que cuando se cree un nuevo proceso o se modifique uno existente, el resto de procesos no se vean afectados y se mantengan operativos en todo momento.

Para realizar el despliegue en caliente se realizan tres operaciones: 1) crear un fichero en el servidor que almacena todos los documentos relevantes para el completo despliegue del proceso; 2) copiar en el fichero creado todos los WSDLs y XSDs utilizados por los servicios Web que son invocados en el proceso; y 3) copiar el documento WS-BPEL que define el proceso, el WSDL asociado a éste y el fichero deploy.xml.

Además, se ha implementado un mecanismo de versionado, que permita que el despliegue de un proceso no borre las versiones anteriores de este proceso. Para ello, cada vez que se despliega un proceso, se crea una carpeta con un número de versión ascendente con el fin de no afectar las instancias de los procesos que actualmente se están ejecutando. En el video de demostración que se encuentra en el DVD se puede encontrar un ejemplo de cómo se realiza el versionado.

Una vez terminado este último paso, el proceso queda totalmente operativo y disponible para su consumo desde cualquier cliente.

Capítulo 4 Caso de estudio

En este capítulo se describe el caso de estudio. Este caso demuestra la utilidad y eficiencia de la herramienta para la transformación de un modelo BPMN a un proceso WS-BPEL que se muestra en el Capítulo 3.

4.1. Descripción General del Caso de Estudio

El caso de estudio elegido consiste en la implementación de una composición de servicios que ofrezca a los usuarios la posibilidad de comprar un paquete de viajes en el que se incluyen vuelos, hoteles, actividades (por ejemplo, buceo, paracaidismo, etc.), coche de alquiler y seguro de viajes.

El principal motivo para elegir este caso ha sido la intensa comunicación que se produce entre las empresas del sector de viajes. En este sector es habitual ver a empresas ofreciendo sus productos a través de otras aplicaciones las cuales pertenecen a terceros, consiguiendo que sus productos puedan llegar a mucha más gente.

Otra característica importante del sector de viajes, es que los procesos requieren cambios continuamente (por ejemplo, cambio de leyes, nuevos servicios, cambios de tendencias, etc.) y estos cambios tienen que poder ser realizados sin parar sus procesos de negocio en ningún momento.

Además, debido a los continuos cambios en el negocio, se necesita una manera de poder reflejar estos cambios de forma fácil en la composición de servicios. Además, es deseable contar con un alto nivel de abstracción para realizar las modificaciones incluso cuando faltan conocimientos de programación.

Como consecuencia de estas necesidades, la venta de paquetes de viajes es un buen caso para poner a prueba la herramienta propuesta.

4.2. Requisitos

En esta sección se explican los requisitos del sistema para la venta de paquetes de viajes. Para guiar la explicación se utiliza un diagrama BPMN que representa de forma simplificada el proceso seguido por el sistema. El modelo simplificado se puede ver en la Figura 51.

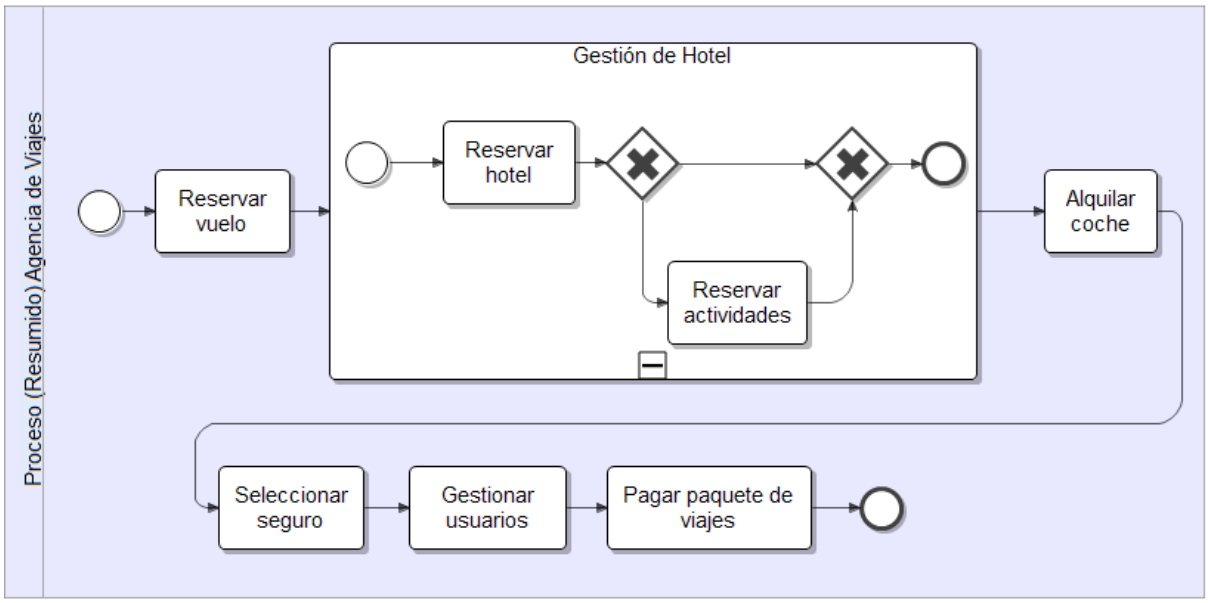

Figura 51: Proceso de negocio simplificado para la venta de paquetes turísticos

A continuación se explican cada una de las actividades del diagrama y su funcionalidad dentro del sistema de venta de paquetes:

Reservar vuelo:

El usuario puede buscar los vuelos disponibles en función de la fecha de llegada, la fecha de salida, el lugar de salida y el destino. Una vez que el usuario ha seleccionado un vuelo, se hace la reserva de ese vuelo.

Gestión de hotel

Esta actividad está formada por varias sub-actividades: 1) Reservar Hotel: permite al usuario la búsqueda de hoteles dado el lugar y la fechas de entrada y salida, y la realización de la reserva del hotel una vez que el usuario lo ha seleccionado. Esta actividad siempre es ejecutada; y 2) Reservar actividad: permite la búsqueda y reserva de actividades dado un hotel. Una actividad puede ser la práctica de algún deporte, visitas turísticas, etc. Las actividades consisten en servicios extra proporcionados por el hotel reservado. La reserva de actividades no siempre está disponible y por lo tanto no siempre se ejecuta.

Alquilar coche

Esta actividad proporciona la capacidad de realizar búsquedas de coches de alquiler dado el lugar y las fechas deseadas, y una vez seleccionado el vehículo, se realiza la reserva.

Seleccionar seguro

Esta actividad muestra al usuario los seguros disponibles para el paquete de viajes, permitiendo seleccionar el que más le convenga.

Gestión de usuarios

Para poder finalizar la compra del paquete de viajes, el usuario debe registrarse. En caso de ya estar registrado, tiene que identificarse con sus datos para poder finalizar la compra.

Pagar paquete de viajes

Cuando ya se han realizado las reservas de los diferentes productos, se ha seleccionado el seguro deseado, y el usuario se ha registrado o identificado, el usuario realiza el pago del paquete de viajes. Para ello el sistema proporciona al usuario la posibilidad de pagar mediante tarjeta de crédito (VISA y MasterCard).

4.3. Modelo BPMN de la agencia de viajes

El diagrama BPMN mostrado en la Figura 52 muestra el proceso final seguido por la agencia de viajes. Debido a su tamaño, se ha mostrado con todos sus componentes comprimidos. Éstos se pueden ver desplegados en el DVD anexo a este documento en la carpeta de información adicional.

En general, cada una de las actividades compuestas (por ejemplo, ReltalStar, KingInsurance, etc.) representa la invocación de un servicio Web diferente, y dentro de cada una de ellas se invoca a las operaciones que contiene el servicio Web. No obstante, las actividades encargadas del pago del paquete de viajes (tales como, Payment_KI, Payment_RS) sólo están compuestas de la llamada a la operación de pago de los servicios Web previamente involucrados (no representa a un nuevo servicio Web).

La única actividad simple del proceso que es invocada en todos los caminos del proceso es “Show_total_price”. Una actividad simple, a diferencia de las actividades compuestas, implementa una única operación. Además la operación “Show_total_price” es implementada por el propio proceso WS-BPEL, es decir, no invoca a un servicio Web externo.

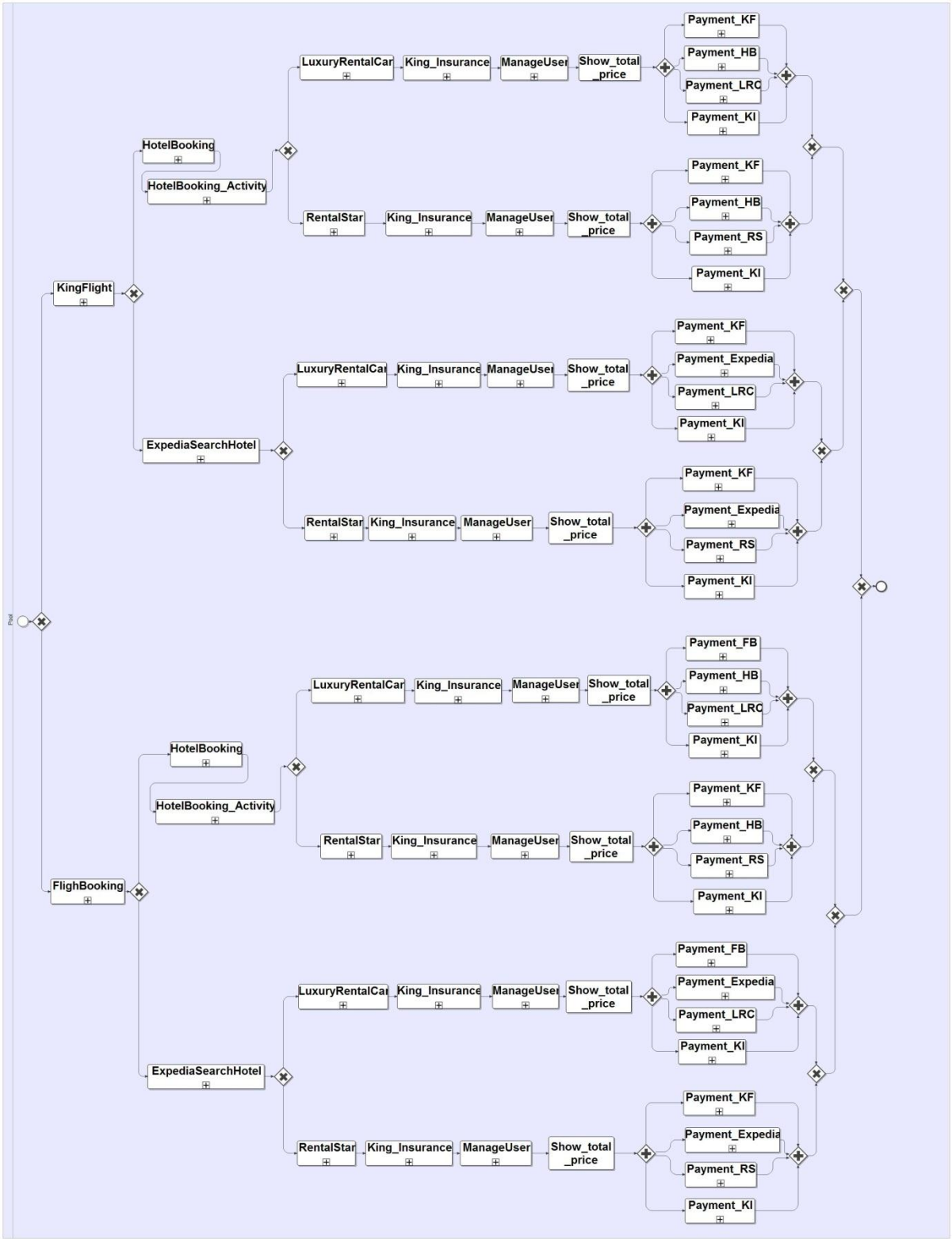

Figura 52: Composición de servicios para la venta de paquetes turísticos

A continuación, en la Tabla 2 se explica cada uno de los servicios compuestos mostrados en la Figura 52. En el DVD anexo a este documento, en la carpeta de información adicional, se encuentra la especificación completa de los servicios Web y sus operaciones.

KingFlight

Esta actividad se encarga de realizar la búsqueda de los vuelos disponibles dados los parámetros introducidos por el usuario y una vez seleccionado, realiza la reserva del mismo.

Operaciones

Search_Flight: Busca vuelos.

Book_Flight: Reserva un vuelo.

Payment_KF

Esta actividad se encarga de realizar el pago de la reserva de un vuelo. Sólo es invocada si previamente ha sido utilizado el servicio Web KingFlight para realizar la reserva de un vuelo.

Operaciones

Do_payment: Realiza el pago de la reserva del vuelo.

FlightBooking

Esta actividad se encarga de realizar la búsqueda de los vuelos disponibles dado los parámetros introducidos por el usuario y una vez seleccionado, realiza la reserva del mismo.

Operaciones

Look_for_flight: Busca vuelos.
Flight_booking: Reserva un vuelo.

Payment_FB

Esta actividad se encarga de realizar el pago de la reserva de un vuelo. Solo es invocada si previamente ha sido utilizado el servicio Web FlightBooking para realizar la reserva de un vuelo.

Operaciones

Do_Payment: Realiza el pago de la reserva del vuelo.

HotelBooking

Esta actividad se encarga de realizar la búsqueda de los hoteles disponibles dado los parámetros introducidos por el usuario y una vez seleccionado, realiza la reserva del mismo.

Operaciones

Searching_Hotel: Busca hoteles.
Hotel_Booking: Reserva un hotel.

HotelBookingActivity

Esta actividad se encarga de realizar la búsqueda de las actividades disponibles en un hotel y una vez seleccionadas, realiza la reserva de las mismas.

Operaciones

Search_Activities: Busca actividades.

Book_Activities: Reserva actividades.

Payment_HB

Esta actividad se encarga de realizar el pago de la reserva de un hotel. Sólo es invocada si previamente ha sido utilizado el servicio Web HotelBooking para realizar la reserva de un

hotel.

Operaciones

Do_Payment: Realiza el pago de la reserva de un hotel.

ExpediaSearchHotel

Esta actividad se encarga de realizar la búsqueda de los hoteles disponibles dados los parámetros introducidos por el usuario y una vez seleccionado, realiza la reserva del mismo.

Operaciones

GetList: Busca hoteles.
GetInformation: Retorna información detallada sobre un hotel concreto.
GetPaymentInformation: Retorna información detallada sobre la forma de pago de un hotel concreto.
GetAvaliable: Realiza la reserva de un hotel.

Payment_Expedia

Esta actividad se encarga de realizar el pago de la reserva de un hotel. Sólo es invocada si previamente ha sido utilizado el servicio Web ExpediaSearchHotel para realizar la reserva de un hotel.

Operaciones

Do_Payment: Realiza el pago de la reserva de un hotel.

LuxuryRentalCar

Esta actividad se encarga de realizar la búsqueda de los coches de alquiler disponibles dados los parámetros introducidos por el usuario y una vez seleccionado, realiza la reserva del mismo.

Operaciones

Get_all_type_fuel: Retorna todos los tipos de combustible disponibles en este momento.
Get_all_office: Retorna todas las oficina para realizar el alquiler disponibles dada una serie de parámetros.
Get_all_type_class: Retornar todos los tipos de coches de alquiler disponibles.
Loof_for_cars: Busca coches de alquiler.
Select_car: Reserva un coche de alquiler.

Payment_LRC

Esta actividad se encarga de realizar el pago de la reserva de un coche de alquiler. Solo es invocada si previamente ha sido utilizado el servicio Web LuxuryRentalCar para realizar la reserva de un coche.

Operaciones

Do_Payment: Realiza el pago de la reserva de un coche de alquiler.

RentalStar

Esta actividad se encarga de realizar la búsqueda de los coches de alquiler disponibles dado los parámetros introducidos por el usuario y una vez seleccionado, realiza la reserva del mismo.

Operaciones

Get_all_type_fuel: Retorna todos los tipos de combustible disponibles en este momento.
Get_all_office: Retorna todas las oficina para realizar el alquiler disponibles dada una serie

de parámetros. **Get_all_type_class:** Retornar todos los tipos de coches de alquiler disponibles. **Search_car_rental:** Busca coches de alquiler. **Book_car_rental:** Reserva un coche de alquiler.
Payment_RS
Esta actividad se encarga de realizar el pago de la reserva de un coche de alquiler. Sólo es invocada si previamente ha sido utilizado el servicio Web RentalStar para realizar la reserva de un coche.
Operaciones
Do_Payment: Realiza el pago de la reserva de un coche de alquiler.
KingInsurance
Esta actividad se encarga de realizar la búsqueda de los seguros disponibles para el paquete de viajes.
Operaciones
Search_Insurance: Busca coches de alquiler.
Payment_KI
Esta actividad se encarga de realizar el pago del seguro seleccionado para el paquete de viajes.
Operaciones
Do_Payment: Realiza el pago del seguro.
ManageUser
Esta actividad se encarga de realizar la gestión de los usuarios que utilizan los servicios ofrecidos por el sistema. Para poder realizar el pago y terminar con el proceso de compra del paquete de viajes, el usuario debe estar registrado e identificado.
Operaciones
Registre: Permite al usuario registrarse. **Login:** Permite al usuario identificarse.
Show_total_price
Esta actividad se encarga de calcular el precio total del paquete de viajes reservado por el usuario y de mostrarlo al usuario. Ha diferencia del resto de actividades, ésta es simple, por lo que solo implementa una única operación. Además, como se ha mencionado, esta operación es implementada por el propio proceso WS-BPEL, por lo que no implica la invocación de ningún servicio Web externo.

Tabla 2: Actividades del proceso de negocio para la venta de paquetes turísticos

4.4. Implementación de la propuesta

En este apartado se explican los detalles de implementación de la composición de servicios de la agencia de viajes. En su implementación se encuentran diferentes componentes: 1) los servicios Web que implementan las operaciones; 2) la composición de los servicios Web que proporciona la lógica subyacente; y 3) la forma en la que la agencia de viajes se muestra al exterior.

4.4.1. Servicios Web

Los servicios Web son la base del sistema implementado. En ellos se encuentra toda la funcionalidad ofrecida por la agencia de viajes. El objetivo a la hora de realizar la implementación ha sido realizar un caso de estudio lo más realista posible. Por este motivo, se han reutilizado servicios Web ofrecidos por terceros. No obstante, como la gran mayoría de empresas no permiten utilizar sus servicios sin pagar, se han tenido que implementar servicios Web propios para poder completar la funcionalidad del caso de estudio.

A continuación se describe la implementación de los servicios Web propios y de terceros:

Servicios Web propios

En la Figura 53 se muestra la arquitectura utilizada para la implementación de los servicios Web propios.

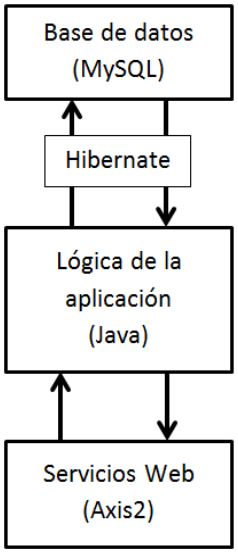

Figura 53: Arquitectura de los servicios Web propios

Todos los servicios Web propios siguen la misma arquitectura. Los detalles sobre cada servicio Web, las operaciones que realizan y los parámetros de entrada y salida de cada operación se pueden ver en la especificación de servicios Web que se encuentra en la carpeta de información adicional del DVD anexo.

Servicios Web de terceros

Los servicios Web de terceros son los servicios ofrecidos por diversas compañías que han podido ser utilizados de forma gratuita para el caso de estudio. Concretamente se ha utilizado la funcionalidad ofrecida por Expedia para la gestión de reservas de hoteles. Toda la documentación sobre los servicios Web de Expedia puede encontrarse en [**54**].

4.4.2. Composición de servicios Web

Los servicios Web descritos en el aparato 4.4.1 ofrecen la funcionalidad para hacer reservas, pagar, buscar vuelos, etc. No obstante, esta funcionalidad no es suficiente. Es necesario establecer un orden en las invocaciones, un control ante errores y adaptar el resultado de las respuestas al cliente.

Para conseguirlo, se ha realizado la composición de servicios Web con WS-BPEL. Con ello se ha conseguido establecer un proceso que se adapta en función de las necesidades del usuario. El código completo WS-BPEL de esta composición se puede encontrar en el DVD anexo, en la carpeta de "WorkspaceAgencyTravel" que esta dentro de la carpeta "Propuesta".

4.4.3. Servicio Web de la venta de paquetes de viajes

Una de las principales ventajas de WS-BPEL es que el resultado de la composición de los servicios Web es otro servicio Web. Es decir, el sistema implementado para la venta de paquetes de viajes es un servicio Web.

Como el sistema es un servicio Web, éste puede ser consumido desde cualquier cliente SOAP. Por ejemplo, para probar la aplicación se ha utilizado el cliente SOAP que proporciona Eclipse por defecto. En la Figura 54 se ve el cliente SOAP de Eclipse.

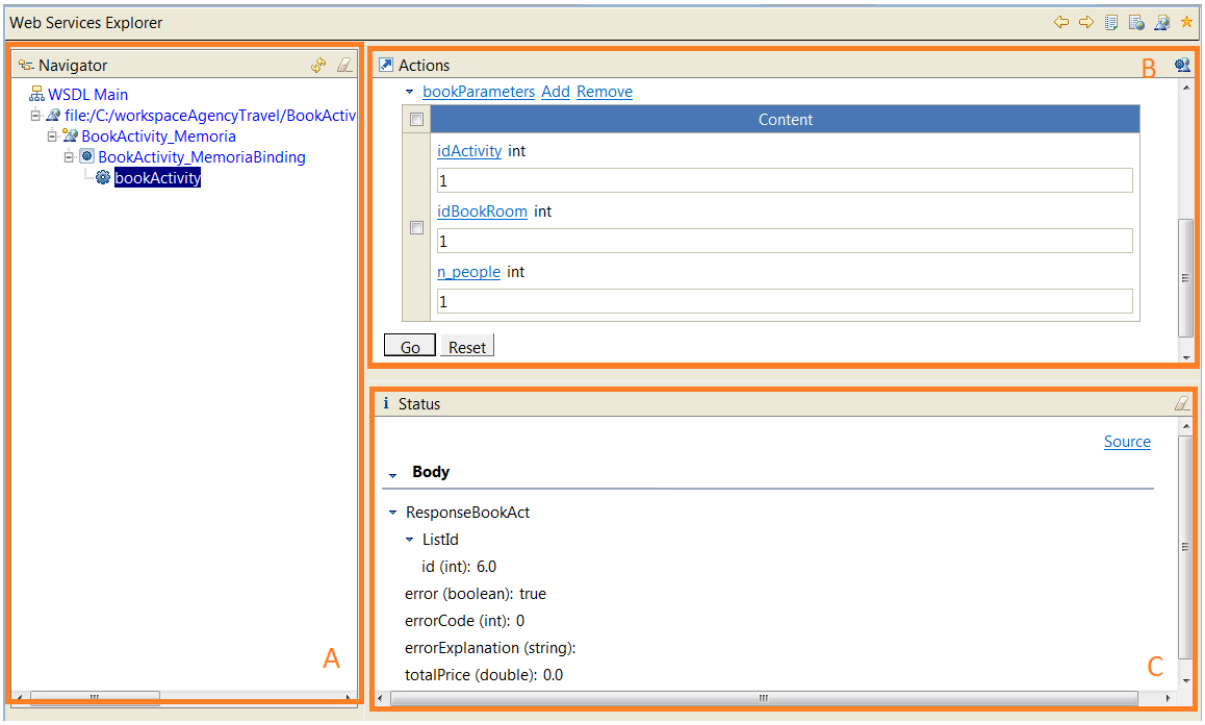

Figura 54: Cliente SOAP de Eclipse

En la Figura 54 se diferencian tres secciones: A) permite navegar por los diferentes elementos del servicio Web, permitiendo ver todas las operaciones que éste contiene; B)

permite especificar los parámetros de entrada de la operación que se quiere ejecutar; y C) muestra el resultado de la ejecución de la operación. También se puede ver el formato XML si se pulsa en el enlace “Source”.

4.5. Arquitectura de la propuesta

En esta sección se explica el entorno donde se ejecuta el caso de estudio, explicando las interacciones entre los diferentes elementos que lo conforman. Para facilitar la comprensión de esta sección se utiliza un diagrama UML de despliegue [**55**] mostrado en la Figura 55.

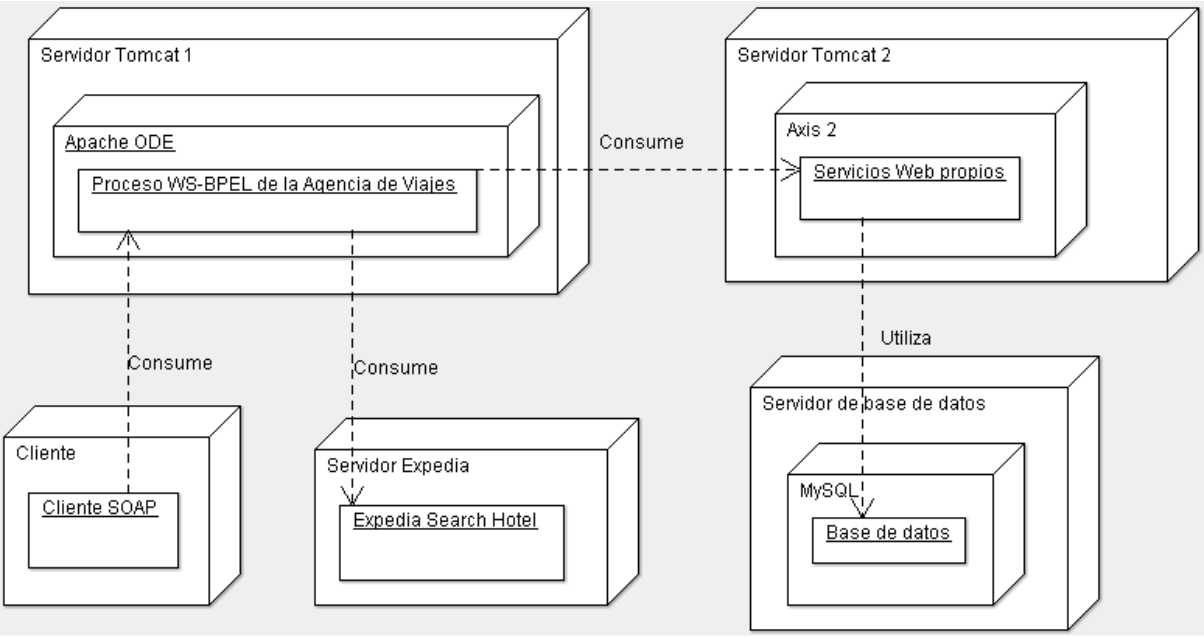

Figura 55: Diagrama UML de despliegue de la agencia de viajes

En la Figura 55 se pueden observar diferentes componentes:

Servidor Tomcat 1

Es el servidor sobre el que se ejecuta el motor de ejecución de procesos escritos en WS-BPEL, que en este caso es Apache ODE.

Apache ODE

Es el motor de ejecución del proceso WS-BPEL encargado de la ejecución del sistema de venta de paquetes de viajes.

Servidor Tomcat 2

Es el servidor utilizado para desplegar los servicios Web en Axis2. Además, la lógica de estos servicios Web se encuentra en dicho servidor.

Axis2

Contenedor de servicios utilizado para el despliegue de los servicios Web propios.

Servidor de base de datos

Contiene al gestor de bases de datos MySQL utilizado para la creación de las bases de datos utilizadas por los servicios Web propios.

Servidor de Expedia

Contiene a los servicios Web de Expedia. Éstos son vistos por el sistema como una caja negra donde se le pasan unos parámetros y se retorna un resultado.

Cliente

Es una representación de uno de los posibles clientes que quiera utilizar los servicios de la agencia de viajes.

Capítulo 5 Trabajos relacionados

En esta sección se describen los proyectos, herramientas e investigaciones relacionadas con la propuesta que se ha implementado. Esta sección esta dividida en varias partes: 1) en primer lugar se explican y clasifican los proyectos que han implementado algún tipo de transformación parcial entre un modelo BPMN y un proceso WS-BPEL; 2) en segundo lugar se explica un proyecto [**56**] que propone una extensión para BPELDesigner basada en BPMN; 3) en tercer lugar se explica una extensión [**57**] sobre el propio lenguaje WS-BPEL para facilitar el completado del WS-BPEL generado a partir de un modelo BPMN; y 4) en último lugar, se habla de la herramienta BPM Petals [**58**] y de su mecanismo para generar código WS-BPEL completo a partir de un modelo BPMN 2.0.

5.1. Proyectos para la generación de WS-BPEL parcial a partir de modelos BPMN

La generación parcial de código WS-BPEL a partir de modelos BPMN es un tema muy trabajado. Muestra de ello es la cantidad de proyecto realizados en este ámbito (ver Tabla 3).

Titulo	Año
A Method of Transition from BPMN to BPEL [**14**]	2011
Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language [**59**]	2011
Interaction Mismatch Discovery Based Transformation from BPMN to BPEL [**60**]	2009
Token analysis of graph-oriented process models [**61**]	2009
Transforming BPMN to BPEL using parsing and attribute evaluation with respect to a hypergraph grammar [**62**]	2009
A Flexible Transformation Scheme between the 'OR' of BPMN and 'Link' of BPEL [**63**]	2008
From business process models to process-oriented software systems [**46**]	2006

Tabla 3: Lista de proyectos para la generación de WS-BPEL parcial

Las transformaciones implementadas por estos proyectos se pueden clasificar en función de tres aspectos: objetivo, implementación de la transformación y restricciones que imponen al usuario con respecto al modelado BPMN. A continuación se explica cada una de estas categorías.

Objetivo

Muchos de [**56**] los proyectos se centran en proporcionar una solución general a la transformación de un modelo BPMN en un proceso WS-BPEL, como es el caso de BABEL y

otros proyectos tales como [**14**] y [**59**]. No obstante, estos proyectos solamente soportan un número limitado de elementos BPMN, imponiendo restricciones al usuario en la manera y en los elementos que puede utilizar para modelar el proceso BPMN.

En cambio, otros proyectos se centran en aspectos muy específicos. Por ejemplo, el proyecto en [**63**] proporciona una solución para poder transformar los *Gateways OR*, o [**60**] que implementa una transformación que permite como entrada varios modelos BPMN que interactúan entre ellos.

Implementación de la transformación

Cada aproximación realiza la transformación del modelo BPMN a código WS-BPEL de manera diferente. No obstante, estas aproximaciones se pueden agrupar en dos bloques: 1) las que utilizan un lenguaje formal o matemático intermedio; y 2) aquellas que utilizan una transformación directa, basada en reglas o en reconocimiento de patrones.

Un ejemplo del uso de lenguajes formales es el proyecto BABEL [**15**], el cual implementa una transformación intermedia del modelo BPMN a Redes de Petri [**47**] antes de generar el código WS-BPEL. Otro ejemplo de ello, es el proyecto [**62**], que transforma un modelo BPMN en un Hypergraph [**64**] para transformarlo posteriormente en el WS-BPEL. En general el uso de una capa intermedia formal permite evitar ambigüedades intrínsecas en BPMN con respecto a WS-BPEL y permite aplicar algoritmos de verificación y de optimización.

Otros proyectos se centran en implementar soluciones lo más sencillas posibles, tratando de realizar el menor número de transformaciones posibles. Con ello se consiguen soluciones más fácilmente modificables. Un ejemplo de este tipo de solución se puede encontrar en el proyecto [**14**] que implementa una transformación basada en patrones. Básicamente, en [**14**] se recorre el modelo BPMN buscando patrones y cada uno de los patrones es transformado dada una serie de reglas en estructuras WS-BPEL.

Restricciones en el modelo BPMN

Muchos de los proyectos ya mencionados establecen restricciones para el usuario que le indican cómo debe modelar el proceso BPMN y qué elementos BPMN puede utilizar. La restricción más habitual está relacionada con el uso de la *Gateways*. Específicamente, muchos proyectos obligan al usuario a que cada *Gateway* que funciona como separador de caminos (“split”), tenga sus correspondiente *Gateway* de unión de caminos (“join”). Este tipo de restricción se puede ver en el proyecto BABEL y en otros proyectos [**62**,**59**].

En cambio, otros proyectos se centran en soportar la mayor cantidad de modelos BPMN, como es el caso del proyecto [**46**], cuyo principal objetivo es solventar el problema de la mayoría de herramientas que limitan la construcción de modelos BPMN mediante restricciones. Con esto se consigue una solución flexible para el usuario.

Para concluir, todos los proyectos explicados hasta el momento, solo soportan la generación de WS-BPEL parcial. Es en este punto es donde nuestra herramienta supera al resto. Específicamente, nuestra herramienta propone una manera para completar el WS-BPEL parcial a partir de una herramienta gráfica basada en formularios. No solo eso, sino que además de completarlo lo despliega en el servidor en caliente y utiliza un control de versiones para garantizar la operatividad de cada proceso.

5.2. Extensión para Eclipse BPEL Designer

En [**56**] se plantea una solución" para la creación de procesos WS-BPEL completos utilizando la notación BPMN. Esta solución a diferencia del resto de proyectos mostrados hasta el momento, no implementa transformaciones. Ésta implementa una representación gráfica para el metamodelo de WS-BPEL basada en BPMN, por lo que cada elemento de WS-BPEL es representado mediante la notación BPMN. Con ello se consigue evitar las diferencias entre la orientación a grafo de BPMN y la orientación estructurada de WS-BPEL.

Para la implementación de esta propuesta, se crea un "plugin" sobre Eclipse BPEL Designer que proporciona una nueva notación gráfica BPMN para representar los procesos WS-BPEL. En la Figura 56 se muestra esta notación gráfica con un ejemplo.

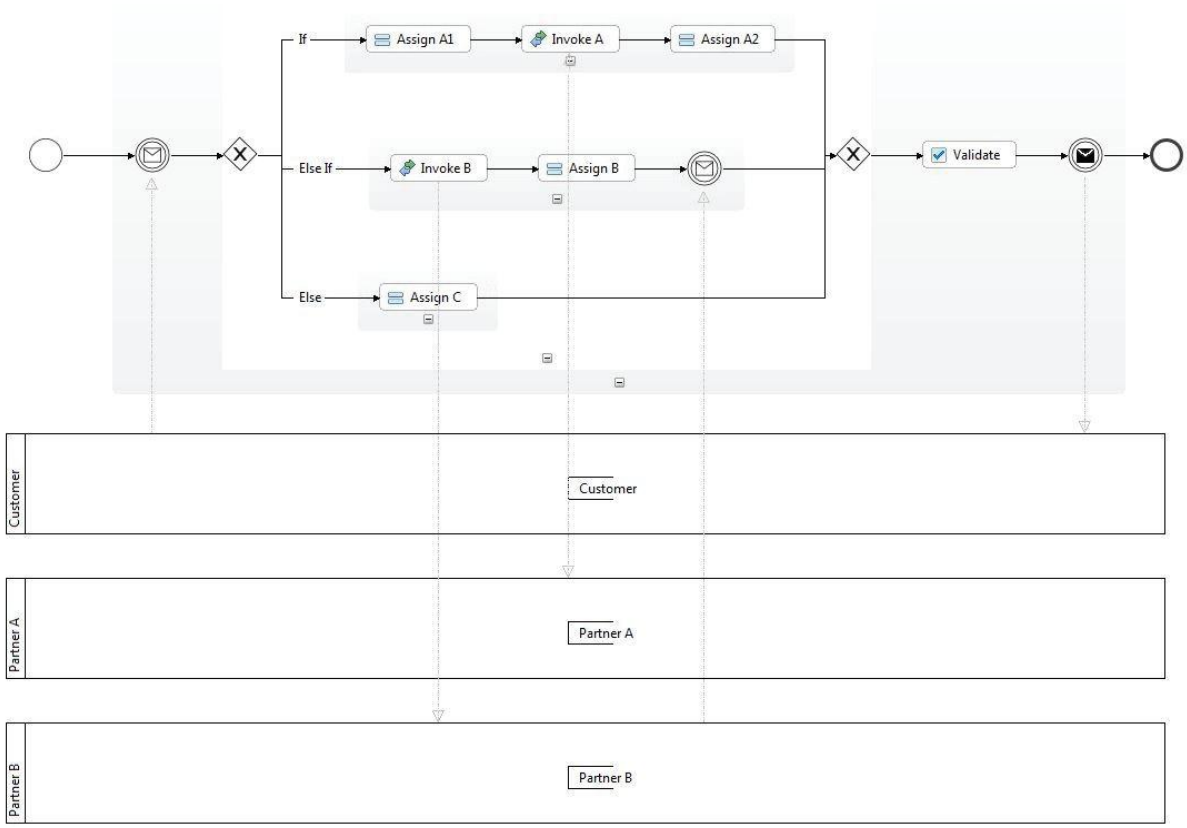

Figura 56: Notación BPMN para Eclipse BPEL Designer [56**]**

No obstante, aunque esta solución permite la creación de procesos WS-BPEL completos, la notación gráfica utilizada no es completamente fiel al estándar BPMN. Esto hace que se pierdan las ventajas inherentes al uso del estándar, y que el usuario tenga que familiarizarse con los nuevos elementos. Por lo que, a pesar de ser una solución ingeniosa para la generación de WS-BPEL completo, no deja de ser similar al propio lenguaje gráfico proporcionado por BPEL Designer. A diferencia de esta solución, nuestra herramienta mantiene la notación estándar proporcionada por BPMN mediante el uso de una herramienta gráfica basada en formularios independiente del modelo BPMN.

5.3. BPELscript

En [**57**] se presenta una solución para la creación de procesos WS-BPEL completos, basada en la generación parcial de código WS-BPEL a partir de modelos BPMN y en el uso de un nuevo lenguaje BPELscript, para completar el código generado.

El proceso consta de los siguientes pasos: 1) se define el modelo BPMN que representa el proceso a generar; 2) se genera la estructura WS-BPEL básica a partir del modelo BPMN; 3) se transforma el código WS-BPEL generado en código BPELscript; 4) el usuario completa los detalles del proceso utilizando BPELscript; y 5) se transforma el código BPELscript en código WS-BPEL.

El principal aporte de esta propuesta es el lenguaje BPELscript, que al ser más similar a un lenguaje de programación clásico, permite al usuario (habitualmente programador) completar el proceso de una manera más sencilla. En la Figura 57 se muestra en el lado izquierdo el proceso WS-BPEL y en el lado izquierdo el equivalente proceso en el lenguaje BPELScript.

Por ejemplo, el control de excepciones en el proceso WS-BPEL se hace mediante la estructura "*<catch> </catch>*" al principio del documento. Esto es muy distinto de como se haría en un lenguaje de programación clásico. Es por ello que en BPELScript esto es transformado en la estructura clásica "*try{} catch()*" que todo programador conoce. Con ello se quiere conseguir acercar la definición de procesos WS-BPEL a la orientación clásica seguida por los lenguajes de programación tales como Java.

Para concluir, aunque la idea de BPELScript puede facilitar la tarea al programador, no supone una gran ventaja para usuarios con pocos conocimientos de programación. Además, es necesario que el usuario aprenda un nuevo lenguaje. Por lo tanto, esta solución esta orientada a un menor grupo de persona que la herramienta explicada en este documento.

Proceso en WS-BPEL

```
<process suppressJoinFailure="yes">
  <faultHandlers>
    <catch faultName="lns:loanProcessFault" >
      <reply operation="request" variable="error"
       faultName="unableToHandleRequest" />
    </catch>
  </faultHandlers>
  <flow>
    <links>
      <link name="receive-to-assess"/>
      <link name="receive-to-approval" />
      ...
    </links>
    <receive operation="request" variable="request"
      createInstance="yes">
      <sources>
        <source linkName="receive-to-assess">
          <transitionCondition>
            $request.amount &lt; 10000
          </transitionCondition>
        </source>
        <source linkName="receive-to-approval">
          <transitionCondition>
            $request.amount &gt;= 10000
          </transitionCondition>
        </source>
      </sources>
    </receive>
    <invoke operation="check"
      inputVariable="request"
      outputVariable="risk">
      <targets>
         <target linkName="receive-to-assess"/>
      </targets>
      <sources>
        <source linkName="assess-to-setMessage">
          <transitionCondition>
            $risk.level='low'
          </transitionCondition>
        </source>
        <source linkName="assess-to-approval">
          <transitionCondition>
            $risk.level!='low'
          </transitionCondition>
        </source>
      </sources>
    </invoke>
    ...
  </flow>
</process>
```

Proceso en BPELscript

```
namespace pns = "http://example.com/loan-approval/";
namespace lns =
  "http://example.com/loan-approval/wsdl/";
@type http://schemas.xmlsoap.org/wsdl/
import lServicePT = lns::loanServicePT.wsdl;
@suppressJoinFailure
process pns::loanApprovalProcess {
partnerLink
  customer = (lns::loanPartnerLT, loanService, null),
  approver = (lns::loanApprovalLT, null, approver),
  assessor = (lns::riskAssessmentLT, null, assessor);
try {
  parallel {
    @portType lns::loanServicePT @createInstance
    request = receive(customer, request);
    signal(receive-to-assess,
      [$request.amount < 10000]);
    signal(receive-to-approval,
      [$request.amount >= 10000]);
  } and {
    join(receive-to-assess);
    @portType lns::riskAssessmentPT
    risk = invoke(assessor, check, request);
    signal(assess-to-setMessage,
      [$risk.level = 'low']);
    signal(assess-to-approval,
      [$risk.level != 'low']);
  } and {
    join(assess-to-setMessage);
    approval.accept = "yes";
    signal(setMessage-to-reply);
  } and {
    join(receive-to-approval, assess-to-approval);
    @portType lns::loanApprovalPT
    approval = invoke(approver, approve, request);
    signal(approval-to-reply);
  } and {
    join(approval-to-reply, setMessage-to-reply);
    @portType lns::loanServicePT
    reply(customer, request, approval);
  }
} @faultMessageType lns::errorMessage
  catch(lns::loanProcessFault) { |error|
    @portType lns::loanServicePT
    @fault unableToHandleRequest
    reply(customer, request, error);
  }
}
```

Figura 57: Comparación entre los lenguajes WS-BPEL y BPELscript [57]

5.4. BPM Petals

BPM Petals es una herramienta en proceso de desarrollo bajo licencia AGPL (Affero General Public License) para el modelado en BPMN 2.0. Ésta proporciona un mecanismo para la generación de procesos WS-BPEL completos a partir de modelos BPMN 2.0 [**58**].

Este proyecto es uno de los más similares a la herramienta para la generación y despliegue de procesos WS-BPEL a partir de modelos BPMN presentada en este documento. Es por ello que se ha decidido hacer una comparación directa entre ambas propuestas:

- Ambas propuestas generan código completo y ejecutable. No obstante nuestra propuesta despliega el proceso WS-BPEL directamente en el servidor y establece mecanismos de versionado para garantizar que los procesos se mantengan 100% operativos en todo momento.

- Petals proporciona los mecanismos para la introducción de la información necesaria para generar un WS-BPEL completo durante la propia definición del BPMN. En cambio nuestra propuesta utiliza una herramienta independiente al modelado BPMN para la introducción de la información faltante. Utilizar una herramienta independiente permite conserva el modelo BPMN intacto (la información de los servicios que implementa el proceso definido en BPMN se encuentra en un modelo independiente). Además la inclusión de información específica del WS-BPEL final mientras se modela el modelo BPMN puede resultar al usuario confuso en comparación a la utilización de formularios proporcionada por nuestra herramienta.

- Petals proporciona soporte para BPMN 2.0 mientras que nuestra propuesta utiliza la versión 1.0 de BPMN. En la actualidad hay muchas empresas que utilizan WS-BPEL para ejecutar sus composiciones de servicios Web. Un salto a BPMN 2.0 les sería costoso y en algunos casos aún impensable. Nuestra propuesta podría apoyar a las empresas que utilizan y seguirán utilizando WS-BPEL.

5.5. Conclusiones

Si bien se ha visto que hay una gran cantidad de herramientas, investigaciones y artículos en el ámbito de la generación de código WS-BPEL a partir de modelo BPMN, pocos se han centrado en el desarrollo de WS-BPEL completo y ejecutable. Las conclusiones obtenidas han sido las siguientes:

- No todo modelo BPMN puede ser transformado en un proceso WS-BPEL ya que la orientación utilizada por BPMN difiere sustancialmente de la utilizada por WS-BPEL. Al no existir una transformación definitiva actualmente o ampliamente aceptada, nuestra herramienta ha sido desarrollada de forma que puede ser utilizada en unión a cualquier transformación de BPMN a WS-BPEL. Esto resulta en reusabilidad y adaptabilidad.

- La no modificación de la notación BPMN es clave si se quieren mantener las ventajas inherentes al estándar. Es por ello que la herramienta se ha implementado separando el modelo BPMN y la información faltante para completar el WS-BPEL. Con ello se consigue una mayor abstracción. Además, tener la información de los servicios Web finales en un modelo independiente permite la rápida generación de diferentes versiones de un mismo proceso BPMN.

Capítulo 6 Conclusión y trabajos futuros

En este capítulo se resumen los aspectos fundamentales de la propuesta, y se muestran los posibles trabajos futuros.

6.1. Conclusión

Mediante la presente investigación se ha dado un paso adelante en la especificación y generación automática de procesos WS-BPEL sobre motores de procesos usando modelos y cumplido los objetivos planteados al principio de este documento.

En primer lugar, se desarrolló una herramienta que transforma un modelo BPMN en código WS-BPEL completo. Esta herramienta se creó mediante: 1) la utilización del proyecto BABEL que genera código WS-BPEL incompleto; 2) creación de una herramienta gráfica que permite al usuario introducir toda la información faltante en el WS-BPEL generado por BABEL en un modelo intermedio; y 3) generación y despliegue de los documentos necesarios para completar el código WS-BPEL a partir del modelo intermedio. Con ello, se ha conseguido elevar el nivel de abstracción en la creación de procesos de negocio basados en WS-BPEL, solventando la complejidad inherente en WS-BPEL y consiguiendo procesos fácilmente modificables y por lo tanto con mayor escalabilidad.

En segundo lugar, se consiguió el despliegue en caliente de un proceso WS-BPEL en el servidor mediante: 1) la incorporación en la herramienta de un sistema de versionado que genera una carpeta diferente cada vez que se despliegue el proceso; y 2) copiando todos los documentos generados en la carpeta generada por el sistema de versionado. Gracias al despliegue en caliente y al versionado de los procesos se ha conseguido que éstos estén operativos en todo momento, que el servidor no tenga que ser reiniciado cada vez que se genera un nuevo proceso y que las modificaciones de un proceso no afecten a los clientes que estén utilizando el proceso en ese mismo momento.

En tercer lugar, para garantizar su futura inclusión en Moskitt y su validez a lo largo del tiempo, se ha construido una herramienta fácilmente adaptable, la cual puede ser añadida a Moskitt y utilizada en unión al resto de extensiones utilizadas por éste.

Por último, se creó y utilizó un caso de estudio realista para demostrar la utilidad de la propuesta.

6.2. Trabajos futuros

Aunque como se ha podido ver, la propuesta ha conseguido solucionar todos los problemas que nos habíamos planteado y cumplir con todos los objetivos, aún quedan muchas cosas que se podrían hacer.

En primer lugar, como ya se ha comentado a lo largo de este documento, la propuesta aquí mostrada ha sido desarrollada con el objetivo de incorporarse como extensión a Moskitt en unión a otras propuestas que se están desarrollando ahora en el grupo PROS. Por ejemplo, hay una propuesta que está trabajando Pedro J. Santana que permitirá a partir de modelos BPMN extendidos generar las interfaces de usuario de aplicación Web que den soporte a los procesos de negocio especificados. Con este tipo de extensiones y herramientas se pretende complementar la herramienta MOSKitt para dar un completo soporte a la especificación e implementación de procesos de negocio complejos.

Por ello, el siguiente paso de esta propuesta será la incorporación de ésta sobre la herramienta Moskitt y la optimización de la comunicación entre la propuesta y las extensiones que se tienen actualmente o que están siendo incorporadas en Moskitt en este mismo ámbito.

En segundo lugar, la utilización de BPMN 2.0 [**65**] para la creación y ejecución de procesos de negocio, en lugar de WS-BPEL es un tema muy discutido actualmente en la Web [**66**,**67**]. Como se ha visto en la memoria, el lenguaje BPMN aporta grandes ventajas tales como que el usuario definir procesos de negocio de forma gráfica e intuitiva. No obstante, conseguir que un modelo BPMN pueda ser directamente ejecutable sin necesidad de otros lenguajes, ha obligado a aumentar el número de elementos necesarios para la definición de los procesos de negocio en BPMN 2.0. Esto hace que el nuevo estándar BPMN 2.0 sea más complejo que su predecesor y que definir un proceso en BPMN 2.0 directamente ejecutable, tenga una complejidad similar o mayor que definirlo sobre WS-BPEL.

Sin embargo, los nuevos elementos definidos en el estándar BPMN 2.0 pueden resultar interesantes en unión a la herramienta que aquí se ha implementado. Mediante la utilización de BPMN 2.0 se podrían simplificar las transformaciones y reducir aún más la complejidad con la que el usuario introduce la información faltante a través de la herramienta. Por ejemplo, han surgido nuevos elementos que especializan al tipo tarea (tales como, *Receive Task*, *Send Task*, *Service Task*) que permiten diferenciar cuando una tarea de BPMN representa una entrada de información, una invocación a un servicio o una salida de información respectivamente.

En tercer lugar, se propone una ampliación a la herramienta implementada. Ésta consiste en la implementación de una “Time Machine” es decir una herramienta para el control de las diferentes versiones de un proceso de negocio. Ésta permitirá al usuario gestionar las diferentes versiones de un mismo proceso, permitiéndole: 1) seleccionar la versión actual de ejecución; 2) volver a anteriores versiones y 3) llevar un historial de qué versión está en ejecución en cada momento y cuáles procesos están actualmente operativos.

Para la implementación de la “Time machine”, se propone el desarrollo de un “Plugin” para eclipse, incorporado a la herramienta aquí presentada, que lleve un historial

con todos los procesos generados a partir de la herramienta, y las versiones de cada proceso. La unión de la “Time Machine” con la herramienta implementada debe ser invisible para el usuario, facilitando al usuario la gestión de los procesos y versiones generadas.

Siguiendo con la idea de la “Time Machine” se podría añadir cierta inteligencia a ésta para que en función del estado de los servicios (por ejemplo, si está en funcionamiento, no está en funcionamiento, o está saturado con demasiadas llamadas) escogiese la versión más optima. Esta herramienta será un sistema inteligente para la selección de la mejor versión en todo momento. Esto se podría realizar mediante la incorporación de reglas, que dotarán a la “Time Machine” de la capacidad de decidir qué versión elegir, es decir, cómo comportarse en cada situación.

Por último, para dar más valor a la propuesta que se ha mostrado a lo largo de este documento, se podrían realizar pruebas con usuarios para medir el desempeño de ellos usando la herramienta. Con este fin, se medirá el tiempo necesario para la implementación de un proceso WS-BPEL con y sin la herramienta. Esto además de proporcionar validez a la herramienta aquí presentada, permitirá mejorarla, y adaptarla en función de las necesidades de los usuarios.

6.3. Información adicional

Se ha adjuntado a este documento un DVD con información referenciada a lo largo de toda la memoria. Este DVD contiene: 1) el código fuente de la herramienta; 2) un manual de usuario que explica como utilizar y configurar la herramienta; 3) un video en el que se muestra la herramienta en ejecución; 4) la especificación completa de los servicios Web propios utilizados para construir el caso de estudio y 4) varios diagramas e información adicional.

Bibliografía

[1] OASIS. (2006) Reference Model for Service Oriented Architecture 1.0. [Online]. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

[2] R. Heffner, L. Fulton, M. Gilpin, H. Peyret, K. Vollmer, and J. Stone, "Topic Overview: Service-Oriented Architecture," *Forrester, June*, vol. 8, p. 4, 2007.

[3] G. H. Alférez, and V. Pelechano, "Context-Aware Autonomous Web Services in Software Product Lines," in *Proceedings of the 2011 15th International Software Product Line Conference*.: IEEE Computer Society, 2011, pp. 100-109.

[4] F. Leymann, D. Roller, and M. T. Schmidt, "Web services and business process management," *IBM Systems Journal*, vol. 41, no. 2, pp. 198 -211, 2002.

[5] OASIS. (2007) Web Services Business Process Execution Language Version 2.0. [Online]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[6] InfoWorld. BPEL: estandarización de la gestión de procesos. [Online]. http://www.iworld.com.mx/iw_SpecialReport_read.asp?iwid=4135&back=2&HistoryParam=

[7] A. Lapadula, R. Pugliese, and F. Tiezzi, "A formal account of WS-BPEL," in *Coordination Models and Languages*.: Springer, 2008, pp. 199-215.

[8] J. Malek, M. Laroussi, A. Derycke, and H. B. Ghezala, "Model-driven development of context-aware adaptive learning systems," in *Advanced Learning Technologies (ICALT), 2010 IEEE 10th International Conference on*.: IEEE, 2010, pp. 432-434.

[9] OMG. (2008) Business Process Model And Notation (BPMN) 1.1. [Online]. http://www.omg.org/spec/BPMN/1.1/PDF/

[10] Consellería de Infraestructura, Territorio y Medio Ambiente. Moskitt. [Online]. http://www.moskitt.org/moskitt/

[11] Apache Software Foundation. Apache ODE. [Online]. http://ode.apache.org/

[12] Oracle. Oracle BPEL. [Online]. http://www.microsoft.com/biztalk/en/us/default.aspx

[13] Microsoft. Microsoft Biztalk. [Online]. http://www.microsoft.com/biztalk/en/us/default.aspx

[14] M. Lu, Q. Cai, and H. Li, "A method of transition from BPMN to BPEL," in *Advanced Intelligence and Awareness Internet (AIAI 2011), 2011 International Conference on*., 2011,

pp. 372 -375.

[15] Australia Queensland University of Technology Brisbane. BABEL Tools. [Online]. http://www.bpm.scitech.qut.edu.au/research/projects/oldprojects/babel/tools/

[16] A. G. Kleppe, J. Warmer, and W. Bast, "MDA explained: the model driven architecture: practice and promise," *Addison-Wesley Longman Publishing Co., Inc.*, 2003.

[17] Object Manage Group (OMG), *MDA Guide Version 1.0.1*, Joaquin Miller and Jishnu Mukerji, Ed., 2003.

[18] W3C. (2004) Web Services Glossary. [Online]. http://www.w3.org/TR/ws-gloss/

[19] W3C. (2004) Web Services Architecture. [Online]. http://www.w3.org/TR/ws-arch/

[20] V. Pelechano, "Arquitectura orientada a servicios SOA," Universidad Politécnica de Valencia, Valencia, 2012.

[21] Microsoft. (2012) Web Service Definition. [Online]. http://msdn.microsoft.com/en-us/library/ms950421

[22] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web services architecture," *IBM Systems Journal*, no. ISSN: 0018-8670, 2002.

[23] A. Benharref, M. A. Serhani, S. Bouktif, and J.Bentahar, "A managerial community of Web Services for management of communities of Web Services," in *New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on*., 2010.

[24] V. Pelechano, "Servicios Web. Estándares, extensiones y perspectivas de futuro," , 2006.

[25] OASIS. (2004) UDDI Version 3.0.2. [Online]. http://uddi.org/pubs/uddi_v3.htm

[26] W3C. (2001) Web Services Description Language (WSDL) 1.1. [Online]. http://www.w3.org/TR/wsdl

[27] W3C. (2001) SOAP Version 1.2. [Online]. http://www.w3.org/TR/2001/WD-soap12-20010709/

[28] W3C. (2012) Extensible Markup Language (XML). [Online]. http://www.w3.org/XML/

[29] W3C. (2004) Hypertext Transfer Protocol (HTTP/1.1). [Online]. http://www.w3.org/Protocols/rfc2616/rfc2616.html

[30] J. Cid. (2007) Gartner: Relación entre Servicios Web y SOA. [Online].

https://blogs.oracle.com/jaimecid/entry/gartner_webservices_soa#

[31] OMG. CORBA. [Online]. http://www.corba.org/

[32] The Open Group. DCE. [Online]. http://www5.opengroup.org/dce/

[33] A. Barco. SOA y los Servicios Web(I). [Online]. http://arquitecturaorientadaaservicios.blogspot.com.es/2006/06/soa-y-los-servicios-web-i.html

[34] S. White, *BPMN Guía de Referencia y Modelado: Comprendiendo y Utilizando BPMN*, CreateSpace, Ed., Jan. 2010. [Online]. http://www.omg.org/spec/BPMN/2.0/

[35] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley, "The Java™ Language Specification Java SE 7 Edition," 2012.

[36] Eclipse Foundation. (2003) Eclipse Platform Technical Overview. [Online]. http://www.eclipse.org/whitepapers/eclipse-overview.pdf

[37] Eclipse Foundation. Eclipse. [Online]. http://eclipse.org/

[38] J. Hunter. JDOM. [Online]. http://www.jdom.org/

[39] W3C. (2010) XML Path Language (XPath) 2.0 (Second Edition). [Online]. www.w3.org/TR/xpath20/

[40] Eclipse Foundation. Soa Tools Platform Project. [Online]. http://www.eclipse.org/stp/

[41] Eclipse Foundation. Business Process Execution Language Project. [Online]. http://www.eclipse.org/bpel/

[42] Eclipse Foundation. Eclipse Modeling Framework Project. [Online]. http://www.eclipse.org/modeling/emf/

[43] Rational Software. Rational Rose. [Online]. http://www-306.ibm.com/software/rational/

[44] C. S. Rodríguez, and J. J. Fons Cors, "Desarrollo de Editores MOSKitt-FEFEM sobre modelos EMF persistidos en Base de Datos haciendo uso de CDO y FMF," 2011.

[45] Conselleria d'Infraestructures i Transport. Moskitt. [Online]. http://www.moskitt.org/

[46] Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling Chun Ouyang. (2006) From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way. [Online]. http://eprints.qut.edu.au

[47] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, 1989.

[48] D. Jayasinghe, *QuickStart Apache Axis2*.: Packt Publishing Ltd, 2008, 2008.

[49] Apache Software Foundation. (2012, Apr.) Axis 2. [Online]. http://axis.apache.org/axis2/java/core/

[50] Apache Software Foundation. Apache Tomcat. [Online]. http://tomcat.apache.org

[51] JBoss Comunity. Hibernate. [Online]. http://www.hibernate.org

[52] Oracle. Mysql Workbench. [Online]. http://www.mysql.com/products/workbench/

[53] Eclipse Foundation. Atlas Transformation Language. [Online]. http://www.eclipse.org/atl/

[54] ExpediaAfiliate Network. Developer EAN. [Online]. http://developer.ean.com/docs#.UDOUBd3iasg

[55] OMG. (2011, Aug.) OMG Unified Modeling LanguageTM (OMG UML). [Online]. http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/

[56] D. Schumm, D. Karastoyanova, F. Leymann, and J. Nitzsche, "On Visualizing and Modelling BPEL with BPMN," in *Grid and Pervasive Computing Conference, 2009. GPC '09. Workshops at the*., 2009, pp. 80-87.

[57] M. Bischof, O. Kopp, T. V. Lessen, and F. Leymann, "BPELscript: A Simplified Script Syntax for WS-BPEL 2.0," in *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*., 2009, pp. 39-46.

[58] EBM Websourcing company. (2012) BPM Petals. [Online]. http://research.petalslink.org/display/petalsbpm/Presentation+of+Petals+BPM

[59] S. Mazanek, and M. Hanus, "Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language," *J. Vis. Lang. Comput.*, vol. 22, no. 1, pp. 66-89, February 2011.

[60] S. Gong, and J. Xiong, "Interaction Mismatch Discovery Based Transformation from BPMN to BPEL," in *Services Computing, 2009. SCC '09. IEEE International Conference on*., 2009, pp. 292 -299.

[61] M. Götz, S. Roser, F. Lautenbacher, and B. Bauer, "Token analysis of graph-oriented process models," in *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW*

2009. 13t68., 2009, pp. 15-24.

[62] S.Mazanek and M. Minas, "Transforming BPMN to BPEL using parsing and attribute evaluation with respect to a hypergraph grammar," *Accepted as a solution for the synthesis case of the GraBaTs*, 2009.

[63] L. Bai and J. Wei, "A Flexible Transformation Scheme between the 'OR' of BPMN and 'Link' of BPEL," in *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*., 2008, pp. 1300 -1307.

[64] National Institute of Standards and Technology (NIST). Hypergraph. [Online]. http://xlinux.nist.gov/dads/HTML/hypergraph.html

[65] OMG. (2011) Business Process Model and Notation (BPMN 2.0). [Online]. http://www.omg.org/spec/BPMN/2.0

[66] F. Leymann. (2009) BPEL vs BPMN 2.0: Should you care? [Online]. http://leymann.blogspot.com.es/2009/12/bpel-vs-bpmn-20-should-you-care.html

[67] M. Rowley. (2009) BPMN 2.0 with BPEL — the debate is just starting. [Online]. http://www.activevos.com/blog/bpel/bpmn-with-bpel-the-debate-is-just-starting/2009/11/23/

[68] W3Schools. XPATH Tutorial. [Online]. http://www.w3schools.com/xpath/xpath_intro.asp

[69] Oracle. (2012) MySQL 5.6 Reference Manual. [Online]. http://dev.mysql.com/doc/refman/5.6/en/