WS-BPEL

Web Services Business Process Execution Language

M.I. Capel

ETS Ingenierías Informática y
Telecomunicación
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada
Email: manuelcapel@ugr.es

DSBCS

Máster en Ingeniería Informática

Índice

Índice

Índice

Índice

Índice

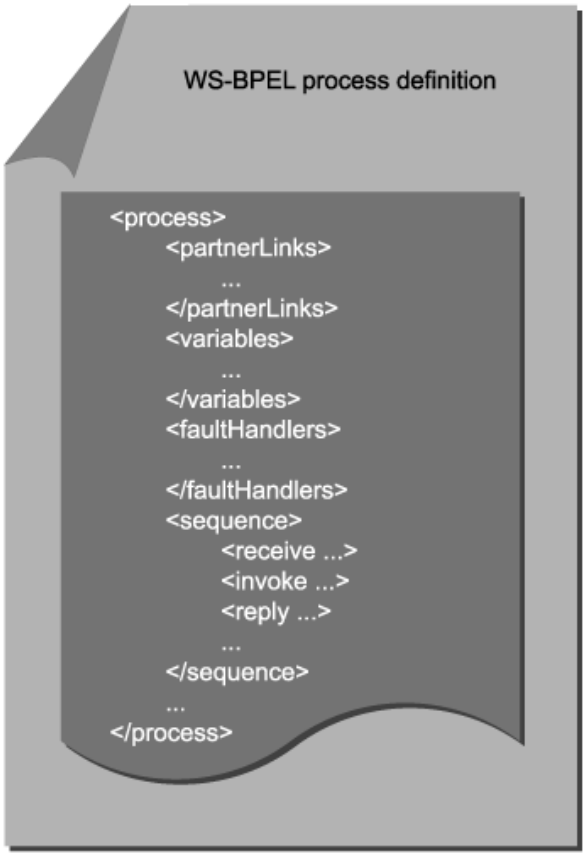
Índice

Índice

Historia de BPEL4WS y WS_BPEL

- Antecedentes: Web Services Flow Language (WSFL) de IBM, especificación XLANG de Microsoft
- Especificación de BPEL4WS 1.0 (Julio, 2002), promovido por IBM, Microsoft y BEA Systems
- BPEL4WS 1.1 (Mayo, 2003) con SAP y Siebel Systems
- Aparición de *motores de orquestación* conformes con BPEL4WS
- Submisión al Comité Técnico de OASIS
- Estándar abierto y oficial de OASIS, que le da un nuevo nombre: WS-BPEL 2.0
- Especificación: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html` (Abril, 2007)

Objetivos



Adquirir una buena comprensión de cómo se puede expresar formalmente un proceso BPEL.

Estructura de la definición de un proceso WS-BPEL común.

El proceso `element`

Elemento raíz de una especificación

Se le asigna un valor de nombre con:

- Atributo `name`
- Establecimiento de los espacios de nombres, relacionados con la definición del proceso

El proceso `element` –sintaxis

```
<bpel:process name="BookstoreABPEL"
    targetNamespace="http://packtpub.com/Bookstore/BookstoreBPEL"
    suppressJoinFailure="yes"
    xmlns:tns="http://packtpub.com/Bookstore/BookstoreABPEL"
    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
        executable">
    <!-- Importar el cliente WSDL -->
    <bpel:import location="BookstoreABPELArtifacts.wsdl"
        namespace="http://packtpub.com/Bookstore/BookstoreABPEL"
                importType="http://schemas.xmlsoap.org/wsdl/" />
    <partnerLinks> ...
    </partnerLinks>
    <variables> ...
    </variables>
    <sequence> ...
    </sequence>
    ...
</process>
```

Los elementos `partnerLinks`

PartnerLink

Define con qué servicios u otros procesos se va acomunicar el proceso *bpel* que queremos definir:

- Lista de servicios participantes en este proceso BPEL.

Características

- Es parecido a una instancia del SW al que nos dirigimos.
- Se corresponde con un `portType` de WSDL para un SW
- Los servicios `partner` actuarán como un cliente del proceso, responsable de invocar el servicio de proceso
- Los servicios `partner` pueden ser invocados por el propio proceso de servicio

Los elementos `partnerLinks` II

```
<!-- PARTNERLINKS
                                                    -->
<!-- Lista de servicios participates en este proceso BPE
                    -->
<bpel:partnerLinks>
        <!-- El rol 'client' representa al solicitante de este
             servicio. -->
        <bpel:partnerLink name="client"
            partnerLinkType="tns:BookstoreABPEL"
            myRole="BookstoreABPELProvider"
         />
    </bpel:partnerLinks>
```

Atributo `partnerRole`

El proceso BPEL entiende o implementa el SW definido en este atributo.

Los elementos `partnerLinks` III

Contenidos de `partnerLink`

- `myRole`: establecer el papel a desempeñar como proveedor de servicio
- `partnerRole`: servicio que proporciona como asociado
- Los atributos `myRole` y `partnerRole` pueden ser utilizados por el mismo elemento `partnerLink`

Ejemplo de construcción `partnerLink`

```
<partnerLinks>
   <partnerLink name="cliente"
      partnerLinkType="tns:TipoEnvioHojaTrabajo"
      myRole="ProveedorServicioEnvioHojaTrabajo"/>
   <partnerLink name="Factura"
      partnerLinkType="inv:TipoFactura"
      partnerRole="ProveedorServicioFacturas"/>
   <partnerLink name="HojaTrabajo"
      partnerLinkType="tst:TipoHojaTrabajo"
      partnerRole="ProveedorServicioHojaTrabajo"/>
   <partnerLink name="Empleado"
      partnerLinkType="emp:TipoEmpleado"
      partnerRole="ProveedorServicioEmpleados"/>
   <partnerLink name="Notificacion"
      partnerLinkType="not:TipoNotificacion"
      partnerRole="ProveedorServicioNotificacion"/>
</partnerLinks>
```

El elemento `partnerLinkType`

Características

- Estos constructos están incrustados dentro de los documentos WSDL de cualquier servicio asociado
- Identificación de los elementos `portType` de WSDL por cada servicio asociado
- Identifican los puertos WSDL referenciados por los elementos `partnerLink` dentro de un proceso

El elemento `partnerLinkType` II

- Múltiples elementos `partnerLink` pueden referenciar el mismo `partnerLinkType`
- 1 papel para cada rol que desempeñe el servicio en los atributos:
  - `myRole` (provee) y
  - `partnerRole` (asociado)

El elemento `partnerLinkType`III

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
DEFINICION DEL TIPO PARTNER LINK TYPE
(en el archivo XXXArtifacts.wsdl)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<plnk:partnerLinkType name="BookstoreABPEL">
   <plnk:role name="BookstoreABPELProvider"
            portType="tns:BookstoreABPEL"/>
 </plnk:partnerLinkType>
```

Variables

Propósito

Sirven para contener datos en BPEL. Cada variable puede contener 1 *valor XSD* o 1 *mensaje WSDL*.

Utilización

Las variables sirven para proporcionar paso de parámetros de entrada o salida en los *endpoint*s de un SW.

El elemento `variables`

La construcción `variables`

- Almacenamiento de información de estado
- Ubicación de *mensajes* completos (`messageType`), de *conjuntos de datos* formateados(`element`), y de *tipos de esquemas* XSD (`type`)
- La información en esta construcción se recupera posteriormente durante la realización del proceso

El elemento `variables` – II

atributo `messageType`

Se define para cada mensaje de entrada y salida procesado por la definición de proceso.
El valor de este atributo es el nombre del mensaje que hay en la definición de proceso asociado.

El elemento `variables` – III

```
<variables>
  <variable name="hola_todos"
            messageType="print:PrintMessage"/>
</variables>
```

La variable `hola_todos` se declara como un contenedor de mensajes WSDL de tipo `print:PrintMessage`

El elemento `variables` – IV

elementos `variable` descendientes utilizados por el proceso "BookstoreABPEL"

```
1  <!-- VARIABLES: Lista de mensages y                    -->
2  <!-- documentos XML usados dentro de este proceso BPEL -->
3  <!-- ================================================= -->
4  <bpel:variables>
5    <!-- Referencia al mensaje pasado como entrada en el inicio
         -->
6    <bpel:variable name="input"
7              messageType="tns:BookstoreABPELRequestMessage"/>
8    <!-- Referencia al mensaje que se devolvera al solicitante -->
9    <bpel:variable name="output"
10             messageType="tns:BookstoreABPELResponseMessage"/>
11 </bpel:variables>
```

El elemento `variables` – V

atributo `type`

Sirve para especificar algún tipo de esquema XSD: `'xsd:string'`, `'xsd:integer'`, etc., de XML

```
1 <variables>
2       <variable name="respuesta" type="xsd:string"/>
3       <variable name="oferta" type="xsd:float"/>
4    </variables>
```

Las funciones `getVariableProperty` y `getVariableData`

`getVariableProperty(nombre variable, nombre propiedad)`

Permite recuperar valores de propiedades globales desde las variables

`getVariableData(nombre variable, nombre parte, camino ubicacion)`

A otras partes de la lógica del proceso, permite el acceso a los datos de información de estado guardados en las variables. Recuperar datos de mensajes desde las variables.

Inicialización de variables en BPEL 2.0

Aunque el siguiente código es válido, sin embargo, algunos motores como Apache ODE no aceptan inicializaciones de variables en línea.

```
<variables>
    <variable name="respuesta" type="xsd:string">
        <from>"NoInteresa"</from>
    </variable>
    <variable name="oferta" type="xsd:float">
        <from>100</from>
    </variable>
</variables>
```

Inicialización de variables en BPEL 2.0 - II

```
<variables>
   <variable name="respuesta" type="xsd:string"/>
   <variable name="oferta" type="xsd:float"/>
</variables>
<sequence>
   <receive createInstance="yes" .../>
...
<assign name="inicializacion">
   <copy>
      <from>100</from>
      <to variable="oferta"/>
   </copy>
   <copy>
      <from>"NoInteresa"</from>
      <to variable="respuesta"/>
   </copy>
</assign>
```

Elementos de actualización y movimiento de datos

Utilidad de los elementos

Para copiar valores entre las variables de un proceso de servicio.
Permiten que se pase información a través de un proceso conforme se desarrolla su ejecución.

Características

`copy` puede procesar sólo parte de un mensaje y copiarlo en 1 variable.
`from, to` pueden contener partes opcionales y atributos de consulta.

El elemento `assign`

Generalidades

- La manipulación de variables se hace en los *endpoints*
- o mediante asignaciones

```
1 <assign>
2   <copy>
3     <from><literal>HolaTodos</literal></from>
4     <to>$hola_todos.valor</to>
```

El elemento `assign` – II

- La sintaxis de las variables sigue las expresiones `XPATH`
- El '.' es el separador de la parte del mensaje WSDL
- Se utiliza un separador para especificar un subelemento dentro de tipos complejos:
  `hola_todos.valor/sub_valor`

Los elementos `copy`, `from` y `to`

Dentro de un `assign`, se indica el tipo del *payload* en la copia a 1 variable de mensaje

```
<bpel:assign validate="no" name="DetermineStock">
        <bpel:copy>
    <bpel:from><bpel:literal>
        <tns:BookDataResponse
        xmlns:tns="http://packtpub.com/Bookstore/BookstoreABPEL"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                <tns:BookISSN>tns:BookISSN</tns:BookISSN>
                <tns:StockQuantity>0</tns:StockQuantity>
                </tns:BookDataResponse>
                </bpel:literal>
        </bpel:from>
    <bpel:to variable="output" part="payload"></bpel:to>
    </bpel:copy>

....
```

Los elementos `copy`, `from` y `to`–II

Dentro de un `assign`, se copian los contenidos una variable y de una expresión a 2 variables de mensaje distintas

```
1 ... <bpel:copy>
2 <bpel:from expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:
      sublang:xpath1.0">
3     <![CDATA[number(5)]]>
4     </bpel:from>
5     <bpel:to part="payload" variable="output">
6     <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
          sublang:xpath1.0">
7     <![CDATA[tns:StockQuantity]]></bpel:query>
8     </bpel:to></bpel:copy>
9 ...
```

Los elementos `copy`, `from` y `to`–III

Dentro de un `assign`, se copian los contenidos una variable y de una expresión a 2 variables de mensaje distintas

```
1  ... <bpel:copy>
2          <bpel:from part="payload" variable="input">
3        <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
             sublang:xpath1.0">
4            <![CDATA[tns:BookISSN]]></bpel:query>
5      </bpel:from>
6      <bpel:to part="payload" variable="output">
7            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel
               :2.0:sublang:xpath1.0">
8        <![CDATA[tns:BookISSN]]>
9        </bpel:query>
10     </bpel:to>
11    </bpel:copy>
12 </bpel:assign>
```

El elemento `sequence`

Construcción secuencial

Organización de una serie de actividades que son ejecutadas en un orden secuencial y predefinido

El elemento `sequence` - II

Un *esqueleto* de construcción `sequence` que contiene sólo algunos de los elementos de WS-BPEL

```
1  <sequence>
2      <receive> ...
3      </receive>
4      <assign>
5          ...
6      </assign>
7      <invoke>
8          ...
9      </invoke>
10     <reply>
11         ...
12     </reply>
13 </sequence>
```

El elemento `receive`

Especificación del servicio provisto

La información que un proceso de servicio espera obtener de un cliente externo que le hace una petición

El elemento `receive` II

Atributos del elemento `receive`

`partnerlink`	El servicio del cliente asociado a través de su partnerLink correspondiente
`portType`	Elemento `portType` del servicio del proceso que espera recibir la petición
`operation`	Operación del servicio del proceso que recibe la petición
`variable`	Aquí se guarda el mensaje de petición entrante
`createInstance`	Si el valor es “yes”, la recepción de una petición creará una nueva instancia del proceso

El elemento `receive` III

El elemento `receive` utilizado en el proceso "BookstoreABPEL" para describir el servicio del cliente asociado que ocasiona el arranque del citado proceso

```
1 <!-- Recibir la entrada del solcitante.
2 Nota: Esto mapea a la operacion definida en BookstoreABPEL.wsdl
      -->
3 <bpel:receive name="receiveInput" partnerLink="client"
4               portType="tns:BookstoreABPEL"
5              operation="getBookData" variable="input"
6     createInstance="yes"/>
```

Contraparte del elemento `receive`

Elemento `reply`

Este elemento es responsable de los detalles necesarios para devolver un mensaje de respuesta al servicio de cliente asociado que realiza la petición.
Repite casi todos los atributos del elemento `receive` correspondiente porque comparte el mismo `partnerLink`.

El elemento `reply` II

Atributos del elemento `reply`

`partnerlink`	El mismo `partnerLink` esteblecido en el elemento `receive`
`portType`	El mismo `portType` del elemento `receive` recibir la petición
`operation`	El mismo elemento `operation` del elemento `receive`
`variable`	El elemento `variable` del proceso de servicio que contiene el mensaje a devolver al servicio asociado
`messageExchange`	Permite al `reply` estar asociado a una actividad capaz de recibir un mensaje

El elemento `reply` III

Un elemento `reply` que casa con el elemento `receive` anterior

```
<bpel:reply name="replyOutput"
      partnerLink="client"
         portType="tns:BookstoreABPEL"
        operation="getBookData"
         variable="output"/>
```

Los elementos `switch`, `case` y `otherwise`

Añadir lógica condicional a la definición de los procesos de servicio

El elemento `switch` establece el ámbito de la lógica condicional que se va a definir.
Se pueden anidar múltiples construcciones `case` para comprobar si se cumplen varias condiciones, cada una dependiendo de un atributo `condition`.
El elemento `otherwise` seañade como una claúsula por defecto al final del `switch`.

Los elementos `switch`, `case` y `otherwise` II

Esqueleto de un elemento `case` donde el atributo `condition` utiliza la función `getVariableData`

```
1 <switch>
2    <case condition=
3       "getVariableData('MensajeRespuestaEmpleado',␣'
            ParametroRespuesta')=0">
4       ...
5    </case>
6    <otherwise>
7       ...
8    </otherwise>
9 </switch>
```

El elemento `invoke`

Idea fundamental

La definición de un proceso BPEL no especifica lo que hace exactamente un SW y cómo lo hace.
La información relativa a la definición e implementación de un SW está contenida en su archivo WSDL.

```
1 <invoke partnerLink="servicioImpresion"
2    operation="print"  inputVariable="hola_todos"/>
```

El elemento `invoke` II

Operación de un servicio asociado

`invoke` identifica la operación de un servicio asociado que el proceso pretende invocar durante su ejecución

```
<invoke name="ValidarHorasSemanales"
    partnerLink="Empleado"
    portType="emp:InterfazEmpleado"
    operation="GetLimiteHorasSemanales"
    inputVariable="PeticionHorasEmpleado"
    outputVariable="RespuestaHorasEmpleado"/>
```

El elemento `invoke` III

Atributos del elemento `invoke`

`partnerlink`	Comunica al motor BPEL la dirección del SW que se invoca
`portType`	Identifica el elemento `portType` de un servicio asociado
`operation`	Operación a la que el proceso envía su petición
`inputVariable`	mensaje de entrada, comunicarse con la operación asociada
`outputVariable`	El valor devuelto se guarda en un elemento `variable` separado

Los elementos `faultHandlers`, `catch` y `catchAll`

Características

- Un mensaje de fallo de WSDL o la utilización de la claúsula `throw` pueden provocar los fallos en el servicio
- Una construcción `faultHandlers` puede contener múltiples elementos `catch` para desarrollar actividades de manejo de errores cuando se produzcan.
- Puede terminar con un `catchAll` para contener actividades de manejo de errores por defecto

Los elementos faultHandlers, catch y catchAll

La construcción faultHandlers conteniendo construcciones catch y catchAll

```
1 <faultHandlers>
2     <catch faultName="AlgoOcurrio"
3            faultVariable="FallaHojaTrabajo">
4         ...
5     </catch>
6     <catchAll>
7         ...
8     </catchAll>
9 </faultHandlers>
```

Otros elementos WS-BPEL

Elemento	Descripción
`compensationHandler`	A WS-BPEL process definition can define a compensation process that kicks in a series of activities when certain conditions occur to justify a compensation. These activities are kept in the compensationHandler.
`correlationSets`	WS-BPEL uses this element to implement correlation, primarily to associate messages with process instances. A message can belong to multiple correlationSets. Further, message properties can be defined within WSDL documents.
`empty`	This simple element allows you to state that no activity should occur for a particular condition.

Otros elementos WS-BPEL II

Elemento	Descripción
`eventHandlers`	The eventHandlers element enables a process to respond to events during the execution of process logic. This construct can contain onMessage and onAlarm child elements that trigger process activity upon the arrival of specific types of messages (after a predefined period of time, or at a specific date and time, respectively).
`exit`	See the terminate element description that follows
`flow`	A flow construct allows you to define a series of activities that can occur concurrently and are required to complete after all have finished executing. Dependencies between activities within a flow construct are defined using the child link element.

Otros elementos WS-BPEL III

Elemento	Descripción
`pick`	Similar to the eventHandlers element, this construct also can contain child onMessage and onAlarm elements but is used more to respond to external events for which process execution is suspended.
`scope`	Portions of logic within a process definition can be subdivided into scopes using this construct. This allows you to define variables, faultHandlers, correlationSets, compensationHandler, and eventHandlers elements local to the scope.
`terminate`	This element effectively destroys the process instance. The WS-BPEL 2.0 specification proposes that this element be renamed exit.

Otros elementos WS-BPEL IV

Elemento	Descripción
`throw`	WS-BPEL supports numerous fault conditions. Using the throw element allows you to explicitly trigger a fault state in response to a specific condition.
`wait`	The wait element can be set to introduce an intentional delay within the process. Its value can be a set time or a predefined date.
`while`	This useful element allows you to define a loop. As with the case element, it contains a condition attribute that, as long as it continues resolving to “true”, will continue to execute the activities within the while construct

Servicio Web “Impresión” realizado utilizando la ligadura con Java

Partiendo del proceso BPEL “HolaTodos” que pasa un literal a través de una variable de entrada del servicio “print”, definir con Apache ODE un proceso que haga lo siguiente:

- Operación de impresión que imprima el literal
- Un archivo WSDL que defina:
  1. Cómo utilizar el SW aludido (definir su API)
  2. Cómo está ligado el SW al código Java

Servicio Web “Impresión” realizado utilizando la ligadura con Java - II

Cosas a definir en BPEL 2.0

1. El espacio de nombres *objetivo*
2. Los mensajes WSDL
3. Los tipos de puertos de WSDL
4. Las ligaduras para los tipos de puertos
5. El servicio WSDL
6. Los tipos `PartnerLink`

(1) Espacio de Nombres

Idea general

Los espacios de nombres de XML y `targetNamespace` de BPEL sirven para discriminar los mensajes del mismo nombre pero dirigidos a diferentes SW.

```
1 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2              targetNamespace="http://www.eclipse.org/tptp/
                   choreography/2004/engine/Print"
3              xmlns:tns="http://www.eclipse.org/tptp/
                   choreography/2004/engine/Print"
4              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5              xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
6              xmlns:format="http://schemas.xmlsoap.org/wsdl/
                   formatbinding/"
7              xmlns:java="http://schemas.xmlsoap.org/wsdl/java
                   /"
8              >
```

(2) Mensajes WSDL

Utilidad

La definición de mensajes WSDL especifica cómo han de ser los contenedores para que mantengan los datos de una operación WSDL invocada.

(3) Los tipos de puertos de WSDL

Utilidad

Sirven para describir la API o la interfaz del propio SW.
Representan a una lista de operaciones con parámetros de entrada y salida, cada uno de los cuales es un mensaje WSDL previamente definido.
Cada una de las operaciones puede tener asociados elementos de tratamiento de fallas.

(4) Ligaduras para los Tipos de Puertos WSDL

Idea general

Sirven para especificar cómo se implementa un SW realmente. Describen los que hay en 'otro lado', con el que tratamos cuando requerimos un servicio.

Posibles ligaduras de un SW

- SOAP/HTTP
- Java

(4) Ligaduras para los Tipos de Puertos WSDL II

La operación de un tipo de puerto WSDL se hace corresponder con un método de Java: `print()`.

```
<operation name="print">
    <java:operation methodName="print" parameterOrder="value"/>
</operation>
```

(4) Ligaduras para los Tipos de Puertos WSDL III

El tipo String de XSD se ha hecho corresponder con el tipo de Java `String`.

```
1 <format:typeMapping encoding="Java" style="Java">
2 <format:typeMap typeName="xsd:string" formatType="java.lang.
      String"/>
3 </format:typeMapping>
```

(5) Descripción del servicio con WSDL

Utilidad

Especifica una instancia de un SW, que es implementado utilizando una ligadura concreta y que se encuentra disponible en una dirección determinada.
La dirección es particular de cada ligadura.

(6) Los tipos `PartnerLink`

Idea general

Cumplen con el requisito de BPEL relativo a que toda instancia de un `partnerlink` esté asociada a 1 tipo de puerto WSDL determinado.

roles de un partnerlink

- `partner role`: el proceso BPEL se comunicará con el servicio
- `my role`: los otros clientes se comunican con éste.

"Target" Namespace

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3                 targetNamespace="http://www.eclipse.org/tptp/
                      choreography/2004/engine/Print"
4                 xmlns:tns="http://www.eclipse.org/tptp/
                      choreography/2004/engine/Print"
5                 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6                 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7                 xmlns:format="http://schemas.xmlsoap.org/wsdl/
                      formatbinding/"
8                 xmlns:java="http://schemas.xmlsoap.org/wsdl/java
                      /"
9                 >
```

Los mensajes WSDL

```
1 <!-- engine printout port -->
2     <message name="PrintMessage">
3         <part name="value" type="xsd:string"/>
4     </message>
```

Los tipos de puertos de WSDL

```
<portType name="Print">
        <operation name="print">
            <input message="tns:PrintMessage"/>
        </operation>
    </portType>
```

Las ligaduras para los tipos de puertos

```
1  <binding name="PrintPortWsifBinding" type="tns:Print">
2          <java:binding/>
3
4          <format:typeMapping encoding="Java" style="Java">
5              <format:typeMap typeName="xsd:string" formatType="
                   java.lang.String"/>
6          </format:typeMapping>
7
8          <operation name="print">
9              <java:operation methodName="print" parameterOrder="
                   value"/>
10         </operation>
11     </binding>
```

El servicio WSDL

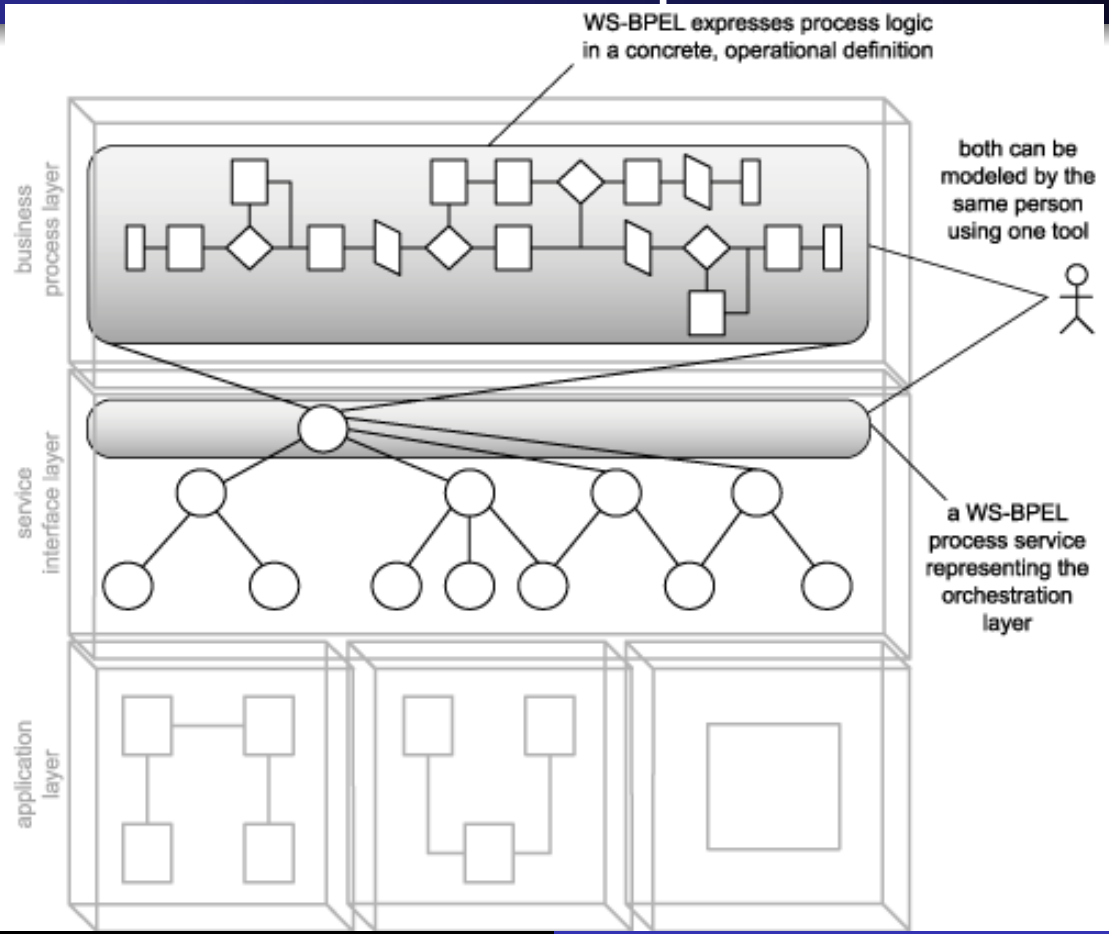
```
1  <service>
2          <port name="JavaPrintPort" binding="tns:
               PrintPortWsifBinding">
3              <java:address className="org.eclipse.tptp.
                   choreography.jengine.internal.extensions.
                   wsdlbinding.wsif.ports.EnginePrinterPort"/>
4          </port>
5      </service>
```

Los tipos `PartnerLink`

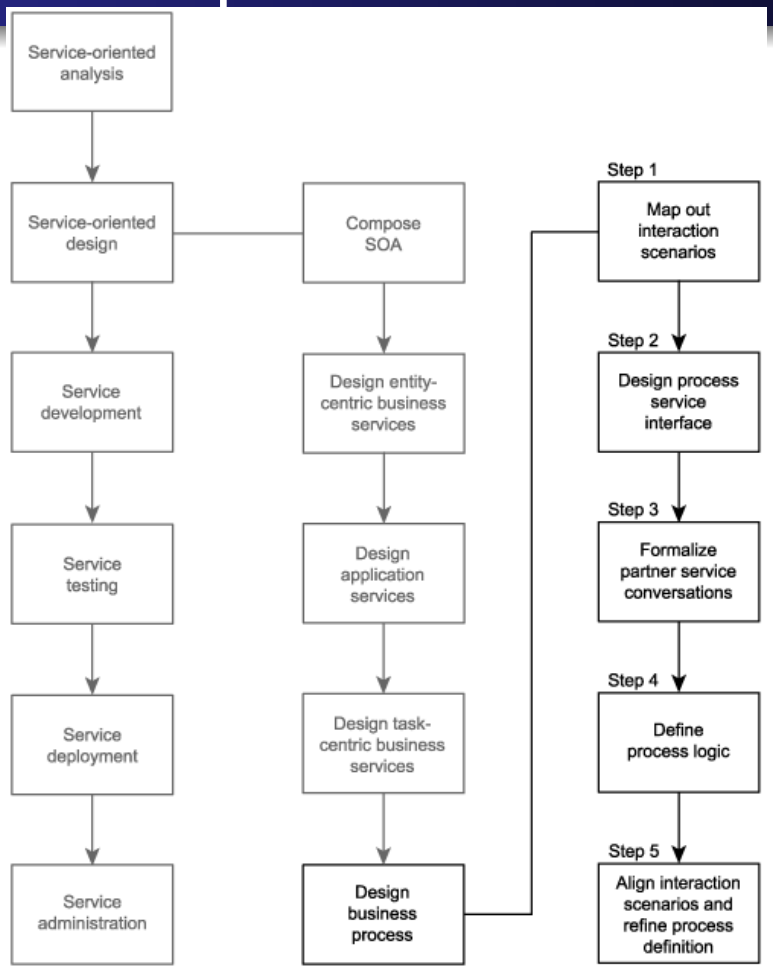
```
<partnerLinkType name="printLink">
        <role name="printService" portType="tns:Print"/>
    </partnerLinkType>

</definitions>
```

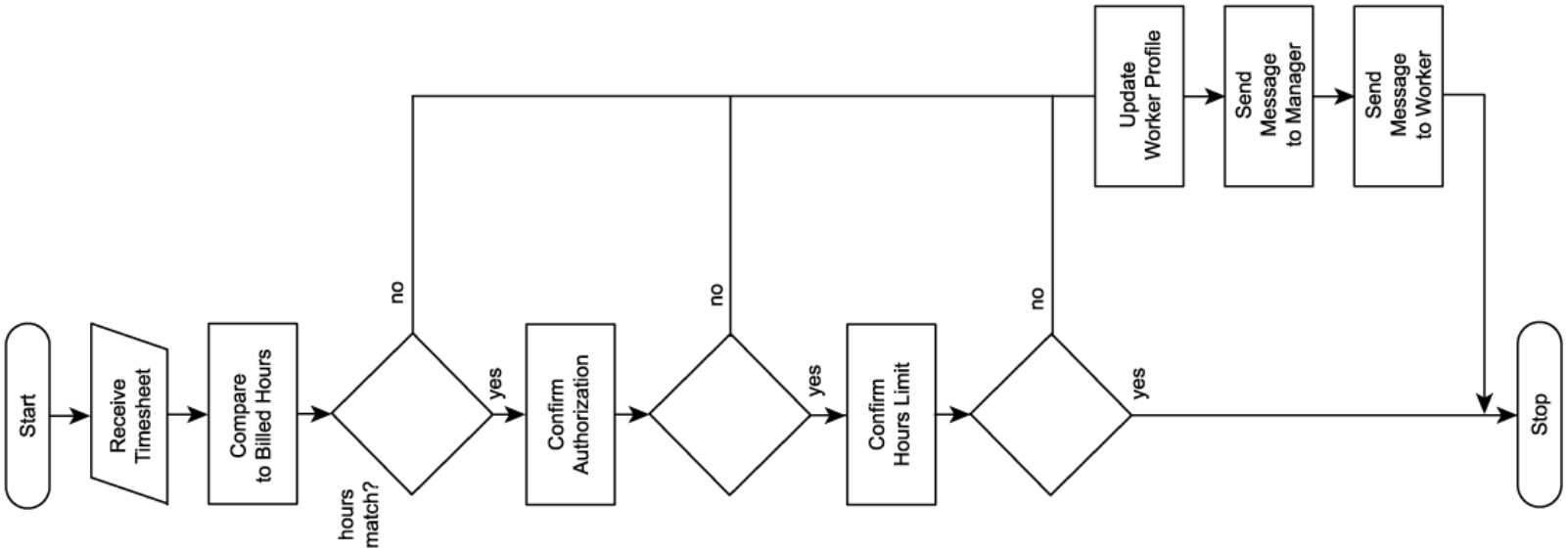
Definición de un proceso de servicio utilizando una herramienta de modelado de procesos



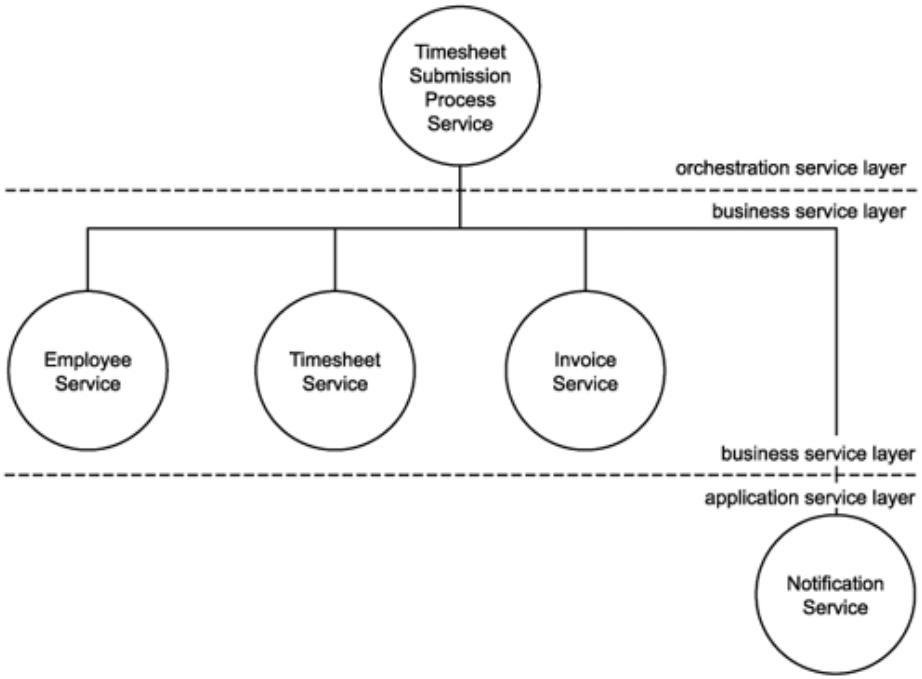
Método de diseño de procesos



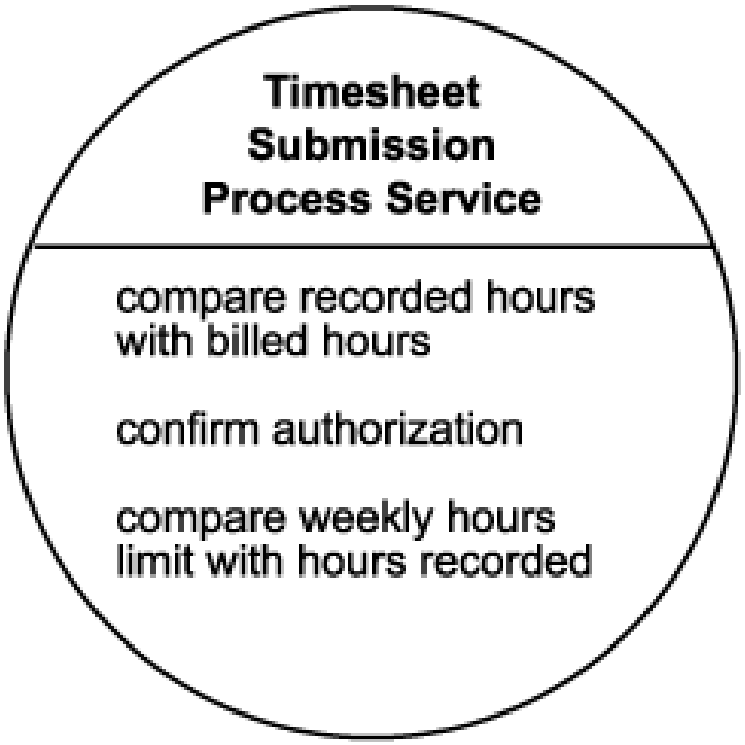
“Proceso Envio Hojastemporización”



Modelo de composición de servicios



Representación abstracta del candidato a "Proceso Servicio Envío Hojastemporización"

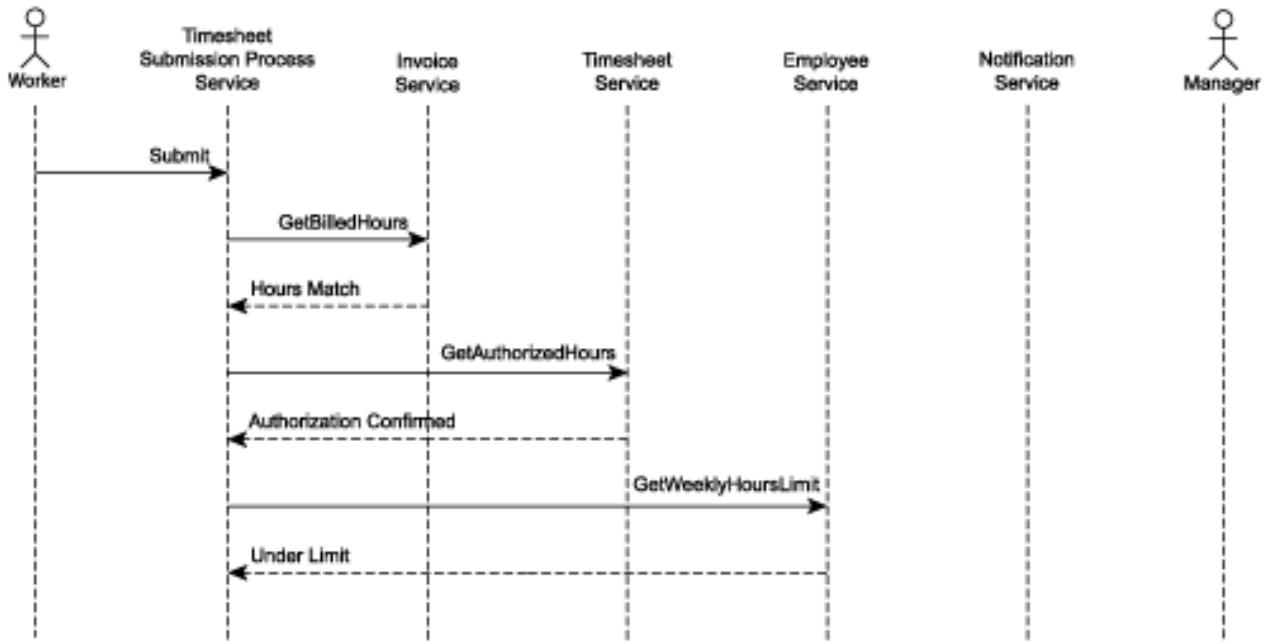


Paso 1: Correspondencia

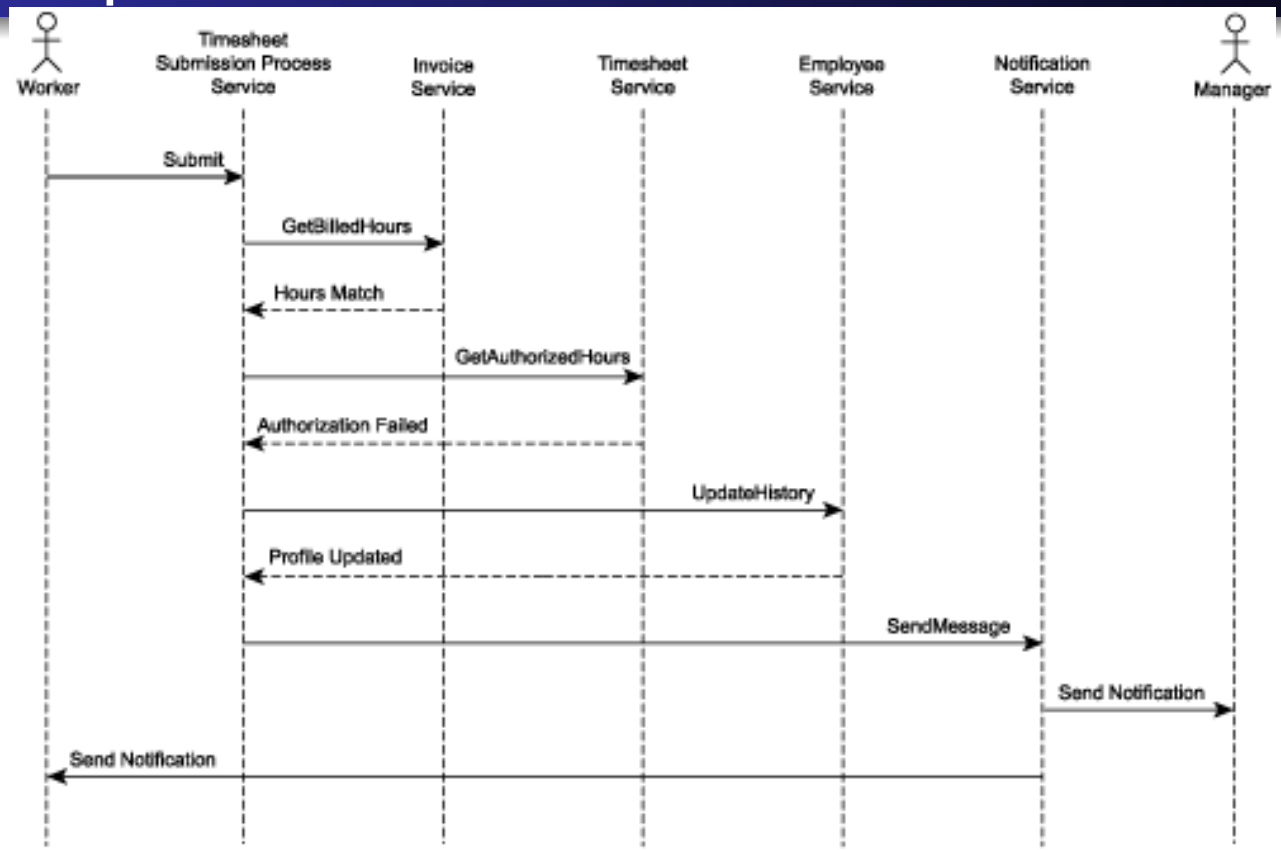
Actividades para definir los requisitos de intercambio de mensajes

- Lógica de flujo de trabajos del proceso de modelado del servicio
- El Proceso de Servicio Candidato
- Otros diseños del Servicio
- *Hacer un mapa* con todos los escenarios posibles de interacción entre el proceso y los asociados al servicio
- Desarrollo de diagramas de actividad UML

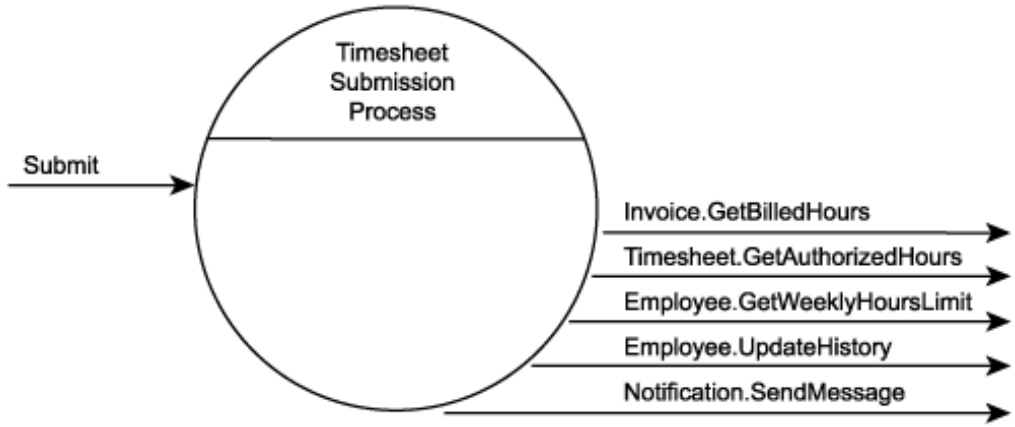
Escenario de interacción entre servicios para completar el Proceso con un envío de la HojaTrabajo válido



Escenario en el que un envío de la HojaTrabajo es rechazado por el Servicio



Los mensajes de petición entrantes y salientes que esperan ser procesados por el “Proceso Servicio Envío HojaTrabajo"



Paso 2: Diseño de la interfaz del proceso de servicio

Actividades a realizar sobre la definición WSDL

- Rellenar la sección `types` con tipos de esquemas XSD necesaris para procesar las operaciones
- Insertando `operation` necesarios, construir la definición WSDL creando el área `portype` o `interface`
- Añadir las construcciones `message` necesarias que contienen elementos `part` que hacen referencia los esquemas apropiados
- Añadir metainformación a través del elemento `documentation`
- Aplicar los estándares de diseño adicionales dentro de los límites de la herramienta de modelado

Definición WSDL del “Proceso Servicio Envío HojaTrabajo”

```
<definitions name="EnvioHojaTrabajo"
    targetNamespace="http://www.xmltc.com/tls/process/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ts="http://www.xmltc.com/tls/timesheet/schema/"
    xmlns:tsd=
        "http://www.xmltc.com/tls/timesheetservice/schema/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.xmltc.com/tls/timesheet/wsdl/"
    xmlns:plnk=
```

Definición WSDL del “Proceso Servicio Envío HojaTrabajo” – II

```
"http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
<types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace=
        "http://www.xmltc.com/tls/
        timesheetsubmissionservice/schema/">
        <xsd:import namespace=
            "http://www.xmltc.com/tls/timesheet/schema/"
            schemaLocation="HojaTrabajo.xsd"/>
        <xsd:element name="Enviar">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="IDContexto"
                        type="xsd:integer"/>
```

Definición WSDL del “Proceso Servicio Envío HojaTrabajo” – III

```
1
2                    <xsd:element name="DocumentoHojaTrabajo"
3                        type="ts:TipoHojaTrabajo"/>
4                </xsd:sequence>
5            </xsd:complexType>
6        </xsd:element>
7    </xsd:schema>
8 </types>
```

Definición WSDL del “Proceso Servicio Envío HojaTrabajo” – IV

```
1  <message name="recibirMensajeEnvio">
2     <part name="CargaMensaje" element="tsd:TipoHojaTrabajo"/>
3  </message>
4  <portType name="InterfazEnvioHojaTrabajo">
5     <documentation>
6        Inicia el Proceso Envio HojaTrabajo.
7     </documentation>
8     <operation name="Enviar">
9        <input message="tns:recibirMensajeEnvio"/>
10    </operation>
11 </portType>
```

Definición WSDL del “Proceso Servicio Envío HojaTrabajo” – V

```
<!--Lo siguiente se anade a cualquier servicio asociado -->
<plnk:partnerLinkType name="TipoEnvioHojaTrabajo">
 <plnk:role name="ServicioEnvioHojaTrabajo">
 <plnk:portType name="tns:InterfazEnvioHojaTrabajo"/>
  </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```

Definición WSDL del “Proceso Servicio Envío HojaTrabajo” VI

```
<!--Lo siguiente se anade a cualquier servicio asociado --->
<plnk:partnerLinkType name="TipoEnvioHojaTrabajo">
 <plnk:role name="ServicioEnvioHojaTrabajo">
 <plnk:portType name="tns:InterfazEnvioHojaTrabajo"/>
  </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```

Paso 3: Formalizar las conversaciones con servicios asociados

Comienzo definición proceso WS-BPEL

- Definir los servicios asociados que participan y asignarles los roles que desempeñan en el intercambio de mensajes
- Añadir las construcciones `partnerLinkType` al final de las definiciones WSDL de cada servicio asociado
- Crear los elementos `partnerLink` para cada servicio asociado dentro de la definición de proceso
- Definir los elementos `variable` para representar los mensajes entrantes y salientes intercambiados con los servicios asociados

Definición del "Servicio Empleado" mejorado con construcciones `partnerLinkType`

```
1  <definitions
2      name="Empleado"
3      targetNamespace="http://www.xmltc.com/tls/employee/wsdl/"
4      xmlns="http://schemas.xmlsoap.org/wsdl/"
5
6    <plnk:partnerLinkType name="TipoEmpleado">
7    <plnk:role name="ServicioEmpleado">
8    <plnk:portType name="tns:InterfazEmpleado"/>
9     </plnk:role>
10    </plnk:partnerLinkType>
11 </definitions>
```

Definición del "Servicio Empleado" mejorado con construcciones `partnerLinkType` – II

Los elementos `partnerLink` para cada servicio asociado

```
1  <partnerLinks>
2     <partnerLink name="cliente"
3        partnerLinkType="bpl:TipoProcesoEnvioHojaTrabajo"
4   myRole="ProveedorServicioProcesoEnvioHojaTrabajo"/>
5     <partnerLink name="Factura"
6        partnerLinkType="inv:TipoFactura"
7        partnerRole="ProveedorServicioFacturas"/>
8     <partnerLink name="HojaTrabajo"
9        partnerLinkType="tst:TipoHojaTrabajo"
10       partnerRole="ProveedorServicioHojaTrabajo"/>
11    ...
```

Definición del "Servicio Empleado" mejorado con construcciones `partnerLinkType` – III

Los elementos `partnerLink` para cada servicio asociado

```
1   ...
2    <partnerLink name="Empleado"
3        partnerLinkType="emp:TipoEmpleado"
4        partnerRole="ProveedorServicioEmpleado"/>
5    <partnerLink name="Notificacion"
6        partnerLinkType="not:TipoNotificacion"
7        partnerRole="ProveedorServicioNotificacion"/>
8 </partnerLinks>
```

Definición del "Servicio Empleado" mejorado con construcciones `partnerLinkType` – IV

Definición de la construcción `variables`

```
<variables>
<variable name="EnvioCliente"
  messageType="bpl:recibirMensajeEnvio"/>
<variable name="PeticionHorasEmpleado"
  messageType="emp:getMensajePeticionHorasSemanales"/>
<variable name="RespuestaHorasEmpleado"
  messageType="emp:getMensajeRespuestaHorasSemanales"/>
<variable name="PeticionHistoriaEmpleado"
  messageType="emp:actualizarMensajePeticionHistoria"/>
<variable name="RespuestaHistoriaEmpleado"
  messageType="emp:actualizarMensajeRespuestaHistoria"/>
    ...
</variables>
```

Definición del "Servicio Empleado" mejorado con construcciones `partnerLinkType` – V

Definición de la construcción `variables`

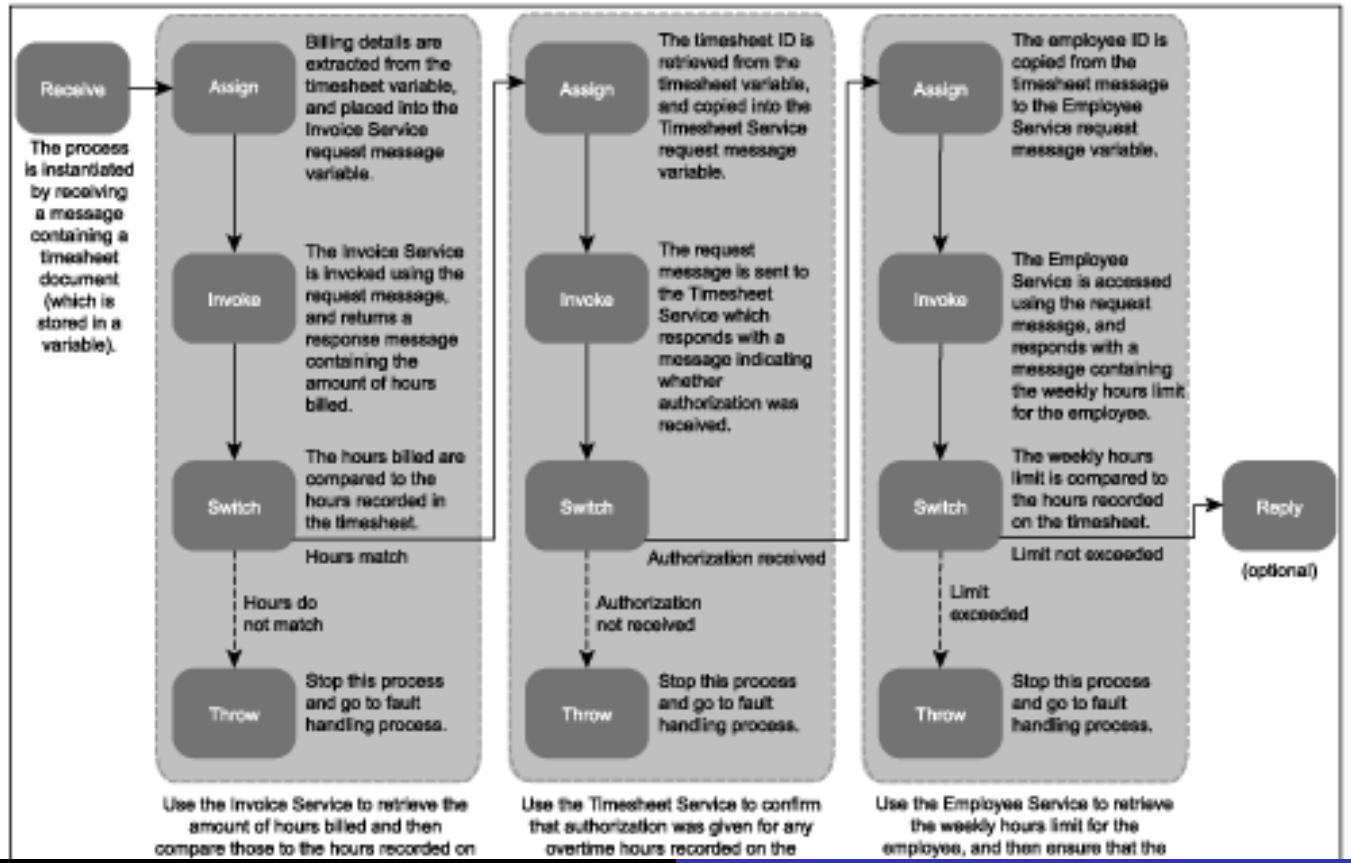
```
1  ...
2    messageType="inv:getMensajePeticionHorasFacturadas"/>
3  <variable name="RespuestaFacturaHoras"
4    messageType="inv:getMensajeRespuestaHorasFacturadas"/>
5  <variable name="PeticionAutorizacionHojaTrabajo"
6    messageType="tst:getMensajePeticionHorasAutorizadas"/>
7  <variable name="RespuestaAutorizacionHojaTrabajo"
8    messageType="tst:getMensajeRespuestaHorasAutorizadas"/>
9  <variable name="PeticionNotificacion"
10   messageType="not:enviarMensaje"/>
11 </variables>
```

Paso 4: Definir la Lógica de Proceso

Proceso de definición

- Toda la inteligencia de flujo de datos hay que convertirla en una definición de Proceso WS-BPEL
- Fijar los elementos `receive` que ofrecen la operación "Envio" a un cliente externo
- Obtener información del cliente utilizando las construcciones `get` y `copy`
- Información al motor para instanciar "Servicio Facturas"
- Comparar *horas trabajadas* con *horas registradas*
- Utilizar la construcción `faultHandlers` para tratamiento de errores si algo falla

Representación gráfica del proceso de la definición de a lógica de proceso



Definición de la Lógica de Proceso – I

Punto de entrada para iniciar el Proceso

```
<receive xmlns=
    "http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    name="recibirEntrada"
    partnerLink="cliente"
    portType="tns:InterfazEnvioHojaTrabajo"
    operation="Envio"
    variable="EnvioAlCliente"
    createInstance="yes"/>
```

Definición de la Lógica de Proceso – II

Asignación de la variable PeticionHorasFacturadas

```
1 <assign name="GetInvoiceID">
2    <copy>
3       <from variable="EnvioCliente" part="cargaMensaje"
4          query="/TipoHojaTrabajo/InfoFacturacion"/>
5       <to variable="PeticionHorasFacturadas"
6          part="ParametroPeticion"/>
7    </copy>
8 </assign>
```

Definición de la Lógica de Proceso – III

Obtención atributos de orquestación para localizar e instanciar el Servicio Facturación

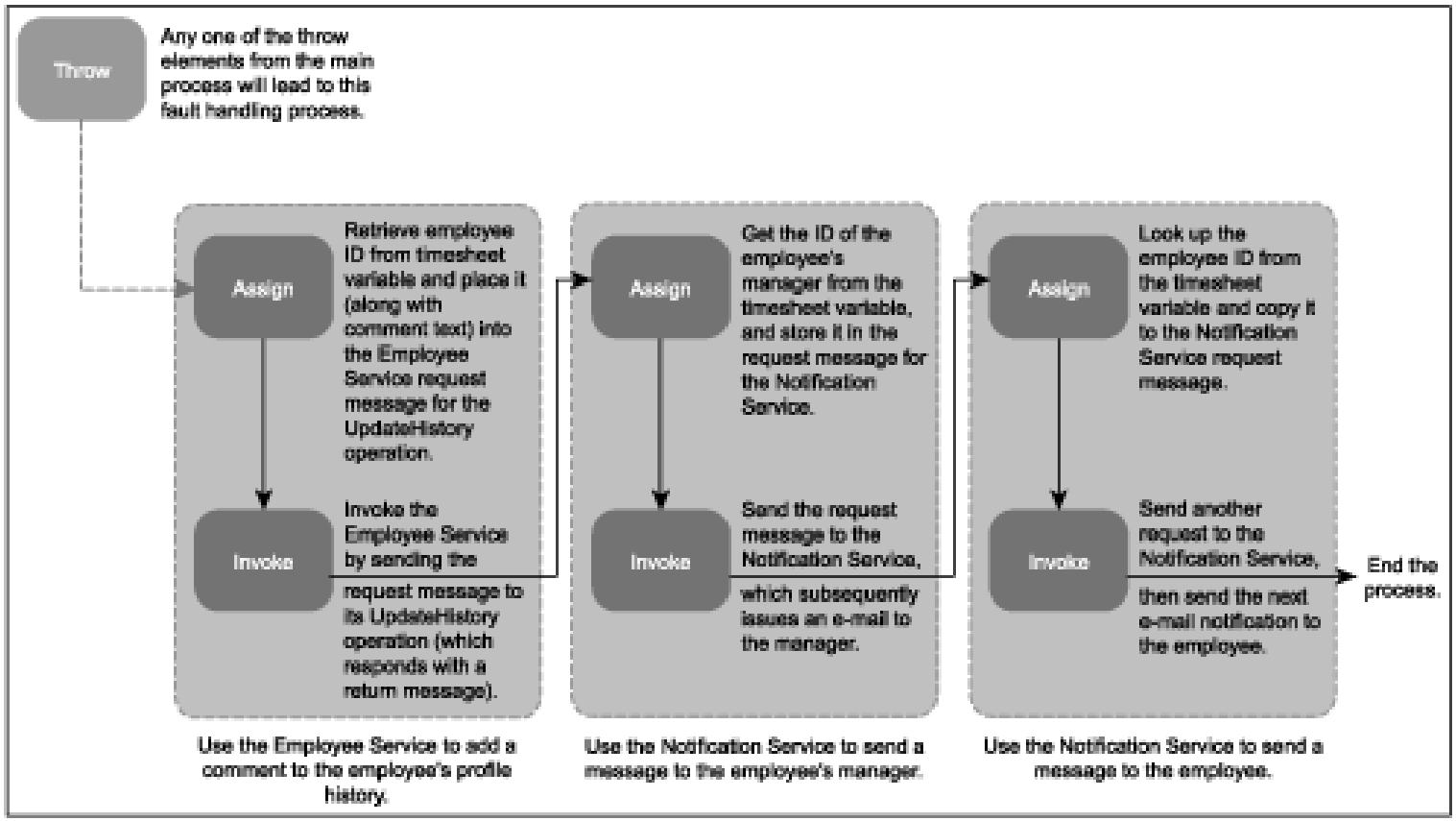
```
<invoke name="ValidarHorasFacturadas"
    partnerLink="Factura"
    operation="GetHorasFacturadas"
    inputVariable="FacturaPeticionHoras"
    outputVariable="RespuestaHorasFacturadas"
    portType="inv:InterfazFactura"/>
```

Definición de la Lógica de Proceso – IV

Implementación de la comparación entre horas facturadas vs. horas registradas

```
1 <switch name="HorasFacturadasConcuerdan">
2    <case condition=
3       "getVariableData('RespuestaHorasFacturadas','
            ParametroRespuesta') != getVariableData('input','
            cargaMensaje', '/tns:TipoHojaTrabajo/Horas/...')">
4       <throw name="ValidacionFallada"
5          faultName="ValidarHorasFacturadasFallado"/>
6    </case>
7 </switch>
```

Representación gráfica de la lógica de proceso dentro de una construcción `faultHandlers`



Ejercicio Propuesto

Especificar la lógica de negocio de los Procesos: *AtencionSanitaria* (actividades de "Ward" en la gráfica BPM) y *EnvíoMedicamentos* (actividades de "Pharmacy") en la gráfica, de manera completa utilizando para ello el lenguaje WS-BPEL introducido en este tema.