1. BPEL
   1. Introducción

BPEL es el acrónimo de Business Process Execution Language. Este es un lenguaje utilizado para la definición de procesos de negocio. En la actualidad BPEL ha llegado a constituirse en la especificación dominante para estandarizar la lógica de integración y la automatización de procesos entre servicios Web [14, 15].

BPEL también puede verse como un lenguaje creado con el objetivo principal de permitir la orquestación, composición y coordinación de servicios Web. Dichos procesos se encuentran compuestos por una serie de actividades que interactúan con otros Web Services, los cuales se denominan partners o colaboradores. Los procesos en BPEL también definidos para que sean Web Services.

En BPEL los procesos pueden ser ejecutables o abstractos. Los procesos abstractos son procesos parcialmente especificados, que deben ser declarados explícitamente como abstractos y que no se crean para ser ejecutados. Estos procesos no incluyen detalles sobre el flujo del proceso. Sólo especifican el intercambio público de mensajes entre diferentes participantes. Por esta razón pueden verse como el contrato de secuenciación de mensajes entre los servicios participantes [8]. Por otra parte, los procesos ejecutables se encuentran completamente especificados y pueden ser ejecutados [19]. Los procesos ejecutables pueden ser sincrónicos o asincrónicos [3]. En un proceso sincrónico se le retorna inmediatamente al cliente la respuesta, y éste se queda bloqueado mientras la ejecución del proceso termina. Por su parte, en los procesos asincrónicos los clientes no se bloquean. En este tipo de procesos eventualmente el proceso realiza un callback y no es necesario que se retorne una respuesta.

En BPEL no se tiene en cuenta la interacción humana. Esto implica que BPEL no proporciona soporte completo como workflow, ni existe el concepto de rol ni de actividad [3]. A este respecto, organizaciones como IBM y SAP han propuesto una extensión a BPEL para la interacción humana (incluyendo conceptos como initiators, approvers, managers, y task orders) en los procesos. Tal extensión recibe el nombre de BPEL4People [3].

* 1. Ejemplo proceso BPEL

En la figura 1 puede apreciarse un proceso BPEL sincrónico, desarrollado empleando *JDeveloper* 10.1.3. El proceso consiste en la consulta de empleos disponibles, utilizando dos *partnerlinks* distintos. La descripción de cómo se ejecuta el proceso se presenta a continuación:

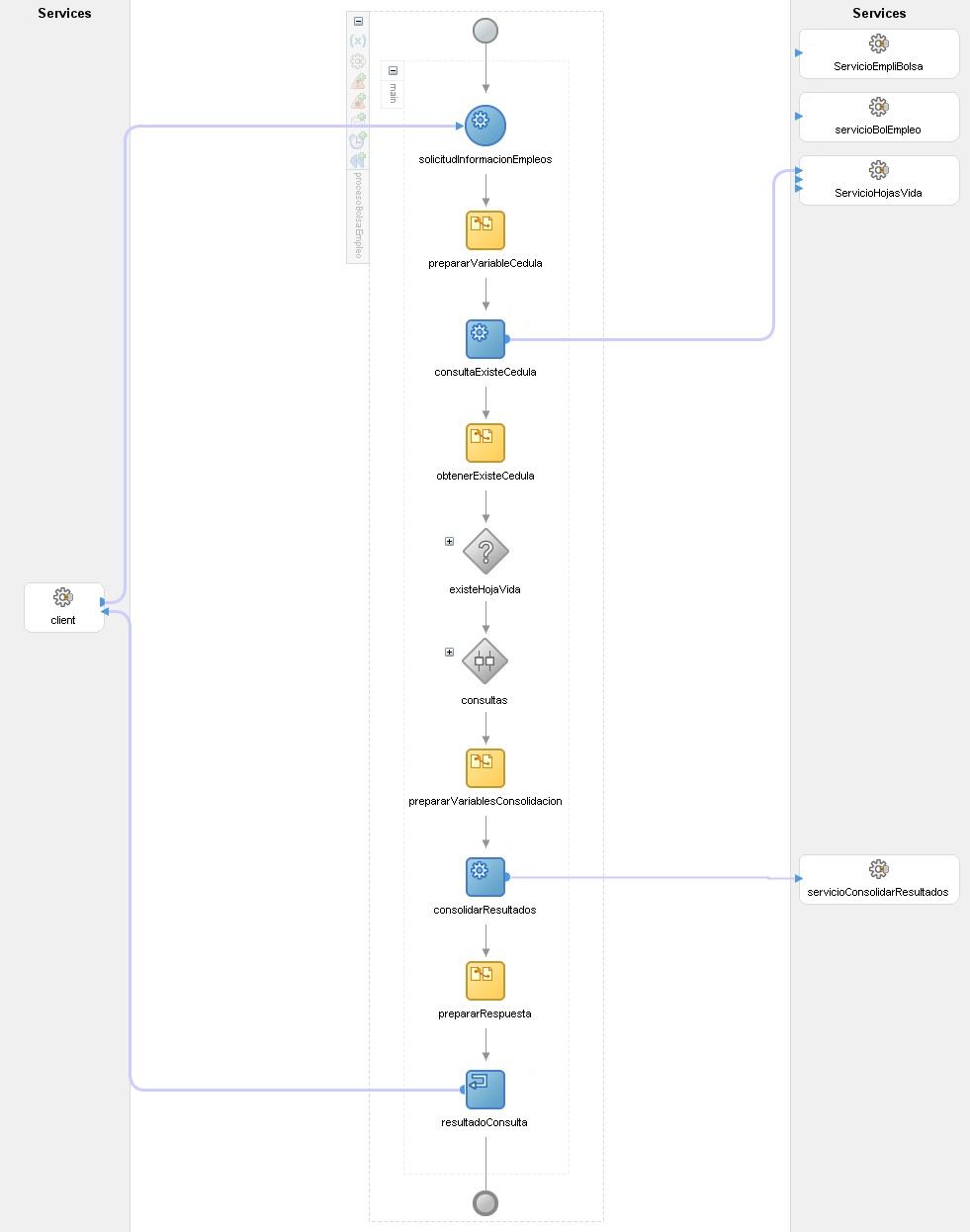


Figure - Proceso de Consulta Empleo

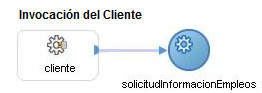
1. **Invocación del proceso por parte de un cliente:** Para la creación del proceso es necesario que éste sea invocado por un cliente (el cual también debe ser un servicio web). Cuando el cliente invoca el proceso, la actividad que recibe el mensaje es “solicitudInformacionEmpleos”, la cual es una actividad *invoke*. La información que el usuario debe especificar son los datos de la hoja de vida (si es la primera vez que ingresa o si se trata de una actualización de datos) o la cédula.

Figure - Proceso Ejemplo BPEL – Invocación del Proceso

1. **Prepara la variable cédula:** Una vez que ya se obtienen los datos del usuario, el siguiente paso es extraer del mensaje de entrada la cédula para preparar el mensaje de entrada del servicio “ServicioHojasVida”. Para ello se utiliza la actividad *assign “*prepararVariableCedula”. Dicho valor será guardado en una variable para luego verificar si el usuario ya había registrado su hoja de vida o no.

Figure - Proceso Ejemplo BPEL- Obtención de la cédula con la actividad assign “prepararVariableCedula”

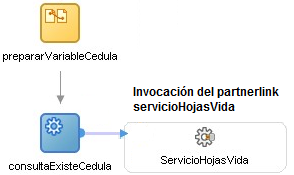
1. **Evaluación Existencia de la Hoja de Vida:** Una vez se tiene la cédula, por medio de la actividad *invoke* sincrónica “consultaExisteCedula” se verifica si ya está registrada. Para ello se utiliza el *partnerLink* “ServicioHojasVida”.

Figure - Proceso Ejemplo BPEL - Evaluación existencia de la hoja de vida usando la actividad invoke consultaExisteCedula

1. **Obtención de la respuesta del *partnerLink* ServicioHojasVida:** Del mismo modo que se hizo con la cédula, es necesario extraer del mensaje de respuesta del *partnerLink* “ServicioHojasVida” si la hoja de vida se encuentra o no registrada. Para ello se emplea la actividad *assign* “obtenerExisteCedula”.

****

Figure - Proceso Ejemplo BPEL- Obtención de la respuesta con la actividad assign “obtenerExisteCedula”

1. **Evaluación registro hoja de vida:** Con la respuesta del servicio “ServicioHojasVida”, por medio de la actividad *if* **“**existeHojaVida” se evalúa si efectivamente la hoja se encuentra registrada.

****

Figure - Verificación de la existencia de la hoja de vida mediante la actividad if “existeHojaVida”.

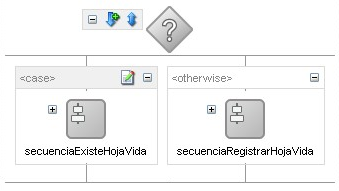


Figure - Actividad if “existeHojaVida” expandida

* 1. Hoja de vida registrada: Si la hoja de vida se encuentra registrada, se ejecutan las actividades contenidas en la actividad *sequence* “secuenciaExisteHojaVida”. Primero se procede a verificar si se trata de una actualización. Para ello se utiliza la actividad *assign* “obtenerEsActualizacion” con el objetivo de obtener del mensaje enviado por el cliente la información que indica si se trata de una actualización o no. Luego con la actividad *if* “actualizacionDatos” se verifica si se debe actualizar la hoja de vida.

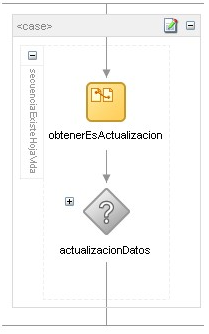


Figure - Expansión actividad sequence “secuenciaExisteHojaVida”.



Figure - Expansión actividad if “actualizacionDatos”

**b)**

* + 1. *Se debe actualizar la hoja de vida*: Si la hoja de vida se debe actualizar, se ejecutan las actividades contenidas en la actividad *sequence* “secuenciaActualizacionDatos”. Primero se utiliza la actividad *assign* “prepararVariableActualizacion” para preparar el mensaje para la actualización. Luego con la actividad *invoke “*actualizarDatos” se invoca de forma sincrónica el *partnerLink* “ServicioHojasVida” para efectuar la actualización.

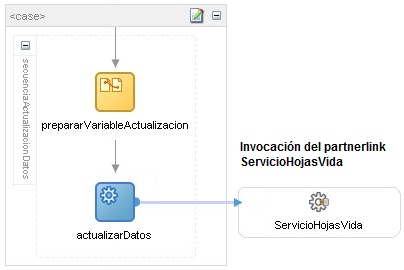
****

Figure - Proceso Ejemplo BPEL- Expansión de la actividad sequence “secuenciaActualizacionDatos”

* + 1. *No se debe actualizar la hoja de vida*: Si la hoja de vida no se debe actualizar, se ejecutan las actividades contenidas en la actividad *sequence* “secuenciaRecuperarInformacion”. Primero se utiliza la actividad *assign obtenerCedula2* para preparar el mensaje requerido para consultar los datos de la hoja de vida del usuario. Entonces, con la actividad *invoke* “recuperarDatosHojaVida” se llama de forma sincrónica el *partnerLink* “ServicioHojasVida”, el cual se encarga de efectuar la consulta. Luego, con la actividad *assign* “asignarDatosRecuperados” se recupera la información del mensaje de respuesta dado por el servicio.

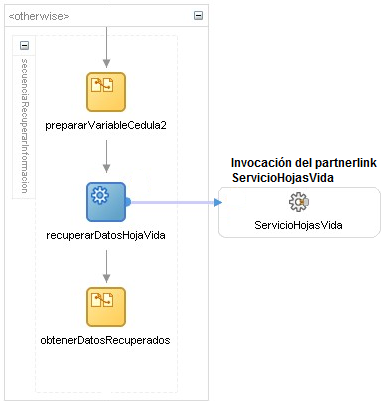


Figure - Expansión de la actividad sequence “secuenciaRecuperarInformacion”

* 1. Hoja de vida no registrada: Si la hoja de vida no está registra, se ejecutan las actividades contenidas en la actividad *sequence* “secuenciaRegistrarHojaVida”. Primero se utiliza la actividad *assign* “prepararVariableCreacion” para alistar el mensaje requerido para registrar la hoja de vida. Luego, con la actividad *invoke* “registrarHojaVida” se llama de forma sincrónica el *partnerLink* *ServicioHojasVida* quién efectúa el registro.

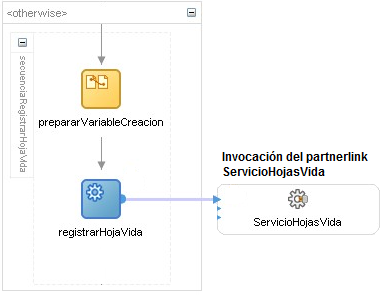


Figure - Proceso Ejemplo BPEL- Expansión de la actividad sequence “secuenciaRegistrar

1. **Consulta de los empleos:** Luego de registrar, actualizar ó recuperar los datos de la hoja de vida, se prosigue con la realización de las consultas. Estas son ejecutadas en paralelo por medio de la actividad *flow* “consultas” e invocando dos servicios diferentes.

****

Figure - Actividad flow “consultas” para la realización de consultas.

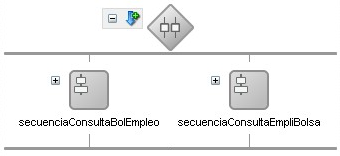
****

Figure - Expansión de la actividad flow “consultas”

* 1. **Consulta ServicioBolEmpleo:** se ejecutan las actividades contenidas en la actividad *sequence* “secuenciaConsultaBolEmpleo”. Para la consulta primero, con la actividad *assign “*prepararVariableConsultaBolEmpleo”, se prepara el mensaje de entrada del servicio “servicioBolEmpleo”. Luego, con la actividad *invoke* “consultaBolEmpleo”, se llama el servicio de forma sincrónica.

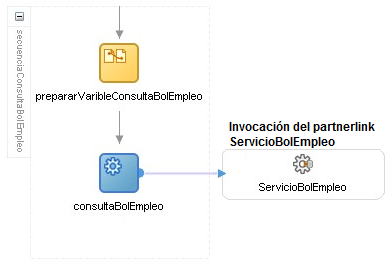
****

Figure - Proceso Ejemplo BPEL- Expansión de la actividad sequence “secuenciaConsultaBolEmpleo”

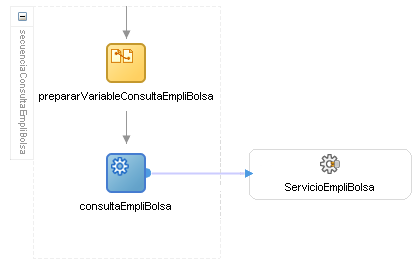
****

Figure - Ejemplo BPEL- Expansión de la actividad sequence “secuenciaConsultaEmpliBolsa”

1. **Preparar mensaje consolidación resultados:** luego de efectuar las dos consultas,por medio de la actividad *assign* “prepararVariablesConsolidacion” se prepara el mensaje de entrada necesario para el servicio que consolida los resultados. Para ello se utilizan los mensajes de respuesta de los servicios de consulta de empleo.

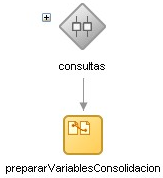
****

Figure - Proceso Ejemplo BPEL- Actividad assign “prepararVariablesConsolidacion” para preparar el mensaje requerido para consolidar los resultados

1. **Consolidar los resultados:** por medio de la actividad *invoke* “consolidarResultados” se invoca el *partnerLink* “servicioConsolidarResultados” de forma sincrónica. Este se encarga de reunir los resultados de los dos servicios de consulta de empleos.

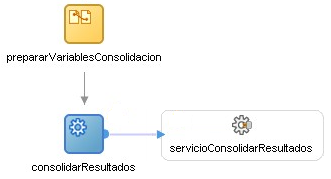
****

Figure - Proceso Ejemplo BPEL- Actividad invoke “servicioConsolidarResultados” para efectuar la consolidación de resultados

1. **Preparar la respuesta:** con la actividad *assign* “prepararRespuesta” se prepara el mensaje de respuesta para enviárselo al cliente.

****

Figure - Proceso Ejemplo BPEL- Actividad assign “prepararRespuesta” para preparar el mensaje de respuesta para el cliente

1. **Enviar la respuesta al cliente:** para terminar el proceso, con la actividad *reply* **“**resultadoConsulta**”** se le envía la lista de empleos al cliente.

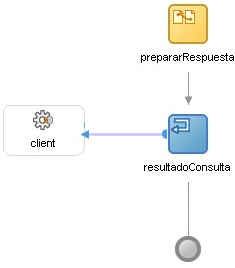
****

Figure - Ejemplo BPEL- Actividad reply “resultadoConsulta” para enviar la respuesta al cliente

* 1. Conceptos básicos
     1. Actividades

Las actividades en un proceso BPEL puden ser clasificadas en dos, actividades básicas y actividades estructurales. Las actividades estructurales permiten establecer patrones de ejecución de las actividades que allí son contenidas. Por ejemplo, la actividad *Sequence* define que las actividades contenidas se deben ejecutar en orden secuencial, *Flow* define ejecución en paralelo, *Switch* define ejecución condicional. Al componer las actividades básicas de esta manera, es posible expresar patrones de control, manejo de fallas y coordinación de mensajes entre instancias de proceso. Las actividades básicas describen los pasos elementales del comportamiento del proceso [1].

* + 1. Partners

Un *Partner Link* representa un canal para una conversación *peer-to-peer* entre el proceso y un partner [2]. Representan las diferentes partes con quien el proceso interactúa, como los clientes y los Web Services que son llamados por el proceso. Un *Partner Link* es una instancia de un conector tipificado entre dos tipos de puerto WSDL, especificando lo que el proceso BPEL provee y que espera del *Partner*.

El modelo de composición de BPEL es tipificado, esto quiere decir que los Web Services son compuestos a nivel del tipo del puerto y no a nivel del puerto. La unión de los Partners a Web Services específicos se hace de acuerdo a diferentes políticas (en tiempo de diseño, en deployment y en tiempo de ejecución) [2].

* + 1. Variables

En BPEL la información se define en variables tipificadas dentro de un XML. Estas variables pueden ser leídas o modificadas por diferentes actividades, ya sea vía mensajes o por manipulación dentro de una actividad. Las actividades de intercambio de mensajes especifican las variables de entrada y de salida que necesitan.

* 1. Conceptos Avanzados
     1. Fault Handling

Los procesos BPEL tienen que tener en cuenta los posibles errores ocurridos por problemas en la invocación de los Web Services o errores que hacen parte de la lógica de negocio del proceso.

Los *Fault* *Handlers* son elementos BPEL que hacen parte de los *Scopes* y su objetivo es atrapar los posibles errores ocurridos. Cuando ocurre un error dentro de un *Scope*, se ejecuta la actividad que está relacionada con el *Fault* *Handler*, esta actividad es responsable de reportar el error o hacer manejo de él. En caso que no se encuentre un *Fault* *Handler* que maneje el error, se enviara el error al *Scope* padre hasta que se encuentre un *handler* o el proceso termine.

* + 1. Compsensation Handling

Debido a que no es posible terminar los procesos largos en una sola transacción atómica, si en algún punto dado ocurre un error, es necesario poder reversar las actividades que fueron ejecutadas satisfactoriamente hasta ese punto. Los *Compensation* *Handlers* son elementos BPEL que permiten deshacer el efecto de las actividades terminadas.

* + 1. Correlation Sets

Debido a que varias instancias del mismo proceso BPEL pueden estar corriendo al mismo tiempo, pero comunicándose con diferentes clientes, se necesita un mecanismo para que comunique a los clientes correctamente con su respectiva instancia. Para esto son los *Correlation* *Sets*.

Estos se definen como propiedades, las cuales son mapeadas a partes de los mensajes que el proceso envía y recibe. Los *Correlations* *Sets* son incorporados a las actividades que se encargan de los mensajes, indicando si la actividad debe iniciar un *Correlation* Set*.* Cuando se recibe un mensaje es posible direccionar dicho mensaje a la instancia de proceso correspondiente.

* + 1. Event Handling

Para soportar eventos asíncronos, BPEL utiliza los *Event* *Handlers*. Estos especifican una actividad que debe ser ejecutada cuando un evento específico ocurre, como un mensaje entrante o el vencimiento de un cronómetro.

* 1. Extensiones en BPEL 2.0

BPEL 2.0 permite que atributos de nombres de espacios calificados aparezcan en cualquier elemento. También permite que elementos de otros espacios de nombres aparezcan dentro de los elementos BPEL. Información adicional sobre la utilidad de estas formas de extensión no es proporcionada por la especificación. Por medio de los elementos *extensions* y *extension* se indican los espacios de nombre de los elementos y atributos que son empleados como extensiones de WS-BPEL

|  |
| --- |
| <process …>  …  <extensions>?  <extension namespace="anyURI" mustUnderstand="yes|no" />+  </extensions>  …  </process> |

Adicionalmente, WS-BPEL proporciona dos elementos para la realización de extensiones:

* *extensionAssignOperation*: Permite la utilización de operaciones para la manipulación de datos, las cuales se encuentran definidas en otros espacios de nombres diferentes al espacio de nombres de WS-BPEL. Este elemento debe especificarse dentro de una actividad *assign*.

|  |
| --- |
| <assign validate="yes|no"? standard-attributes>  …  (<copy keepSrcElementName="yes|no"? ignoreMissingFromData="yes|no"?>  from-spec to-spec  </copy>  |  <extensionAssignOperation>  assign-element-of-other-namespace  </extensionAssignOperation>  )+  </assign> |

* *extensionActivity*: permite la definición de un nuevo tipo de actividad dentro del proceso. La nueva actividad debe tener los elementos y atributos estándar de todas las actividades BPEL. Las actividades de extensión pueden emplearse como cualquier otra actividad en la definición de un proceso BPEL. Esta clase de actividades también pueden contener otras actividades (es decir, las actividades de extensión pueden ser actividades estructuradoras). Las actividades de extensión tienen la siguiente estructura:

|  |
| --- |
| <extensionActivity>  <anyElementQName standard-attributes>  standard-elements  </anyElementQName>  </extensionActivity> |

* 1. Implementaciones

Actualmente se encuentran varias implementaciones de motores libres que son Open Source y comerciales. Dentro de los motores Open Source se encuentran ActiveBPEL [16], Apache ODE [17]. Dentro de las aplicaciones comerciales se encuentran IBM WebSphere Process Server [9], Oracle BPEL Process Manager [10], Microsoft BizTalk Server 2006 [13].

1. Aspect-Oriented Programming

La programación orientada por aspectos (AOP) ha sido utilizada para resolver el problema de modularización de código, debido a preocupaciones transversales. Está metodología es aceptada porque que reduce la cantidad de código enredado y repetido, además de reducir los costos de introducir nuevo comportamiento en la aplicación base. Para lograr esto, AOP propone una nueva unidad de modularidad llamada aspecto, la cual se basa en tres conceptos importantes: join points, pointcuts y advice.

* 1. Conceptos básicos
     1. Joinpoint model

El modelo de join points define los lugares donde los advices van a ser ubicados en la ejecución de la aplicación base. En lenguajes orientados por aspectos, como AspectJ, el modelo de join points está definido por llamados de métodos, acceso a variables, instanciación de objetos, etc.

* + 1. Poincut language

El lenguaje de pointcut es utilizado para seleccionar un conjunto específico de join points. Por ejemplo en AspectJ, el lenguaje de pointcuts provee elementos para seleccionar llamados a métodos, ejecución de métodos y acceso a campos.

* + 1. Advice Language

El lenguaje de advices define la funcionalidad que debe ser ejecutada en join points específicos. El advice es un fragmento de código que es ejecutado cuando se alcanza un join point identificado por el respectivo pointcut. El advice puede ser ejecutado antes, después, o en vez de los join points que son seleccionados por el pointcut.

* + 1. Aspect Weaving

Se necesita un mecanismo para integrar el código existente con el nuevo código definido en los advices. Este mecanismo se conoce como entretejido, y existen dos aproximaciones principales: cuando el entretejido se hace en compilación, entretejido estático, o cuando se hace en ejecución, entretejido dinámico. [6]. En el entretejido estático se pasa por una etapa de transformación del código base, para que éste ahora contenga el código definido en los advices. En el entretejido dinámico, se verifica constantemente si es necesario ejecutar algún advice dependiendo del pointcut que se haya alcanzado.

* 1. Aspect Interference

En la literatura [7], se usan ejemplos de aspectos ortogonales para explicar los beneficios de la programación orientada por aspectos [4]. Sin embargo, no todos los aspectos son ortogonales y por tanto es posible que se presente comportamiento no deseado. Sin embargo, este comportamiento no se da debido a la incorrecta implementación del aspecto como tal, sino a la composición resultante de varios aspectos independientemente programados en un punto especifico. Este comportamiento puede causar conflictos, dependencias o interacciones inesperadas.

Para ilustrar este problema se utiliza un ejemplo definido en [5]. Suponga una aplicación que tiene dos servicios (sendData(String) and recieveData(String)). Debido a cambios en el ambiente o a nuevas reglas de negocio, se necesita programar dos aspectos: uno para monitoreo (LogginAspect) y otro para cifrado (EncryptionAspect). El primer aspecto es colocado antes que comience la ejecución de ambos servicios. El segundo aspecto provee cifrado para todos los mensajes salientes, y descifrado para todos los mensajes entrantes. Esto implica que el aspecto de monitoreo y el de cifrado son colocados en el servicio sendData(String), y el aspecto de monitoreo y descifrado son colocados en el servicio recieveData(String). Ahora considere el servicio sendData(String). Si el aspecto de monitoreo es utilizado para depuración, debería ser aplicado sobre mensajes no cifrados. En este caso es razonable colocar el aspecto de monitoreo antes que el aspecto de cifrado. Sin embargo, suponga que los mensajes intercambiados son considerados tan sensibles de tal manera que cualquier mensaje saliente debería ser cifrado, incluso los de monitoreo, por tanto el aspecto de cifrado debería ser colocado después del aspecto de monitoreo. El orden necesario para colocar los dos aspectos es dependiente del contexto de la aplicación, reglas de negocio, etc. y no puede ser automatizado.

1. Aspect-Oriented Workflow Languages
   1. Introducción y paralelo con AOP

Muchas de las preocupaciones transversales encontradas en las aplicaciones tradicionales, se encuentran en contextos de workflow y como estos sistemas ayudan a resolver problemas específicos, también tienen preocupaciones transversales inherentres. Estás incluyen falta de modularidad [11] y el soporte inadecuado para cambiar la composición del proceso en ejecución. [11].

Usar desarrollo orientado por aspectos para satisfacer estas preocupaciones es propuesto [11, 12], debido a que la orientación por aspectos es un paradigma de descomposición y modularización multipropósito, el cual puede ser aplicado en otros contextos.

* 1. Diferencias con Aspect-Oriented Programming
     1. Crosscutting concerns especificos de workflows – Ejemplo

Un ejemplo de la falta de modularidad en un contexto de workflows, es la necesidad de contar cuantas veces se ejecuta una actividad, tal vez porque cada ejecución representa un costo considerable para la organización. Resolver esta preocupación sin usar la orientación por aspectos implica modificar las instrucciones de dicha actividad. Sin embargo, si la misma lógica es necesaria en otras partes del proceso, la misma lógica debe ser repetida, con la única diferencia del lugar donde es ejecutada. En este caso, si se utiliza orientación por aspectos, el código para contar se define únicamente una vez y luego se coloca en las diferentes partes del proceso donde se necesita.

Un ejemplo del soporte inadecuado a cambios en la composición de un proceso, es agregar una actividad específica a un proceso que ya se encuentra establecido. Por ejemplo, si una nueva regla de negocio indica que es necesario que un supervisor verifique y valide el estado de un proceso en varios puntos (en un proceso bancario o de reclamación de seguros), en los sistemas actuales de workflow implicaría crear una nueva definición de proceso. Para satisface la misma regla de negocio usando lenguajes de workflows orientados por aspectos, una actividad de supervisión puede ser creada y luego colocada en las diferentes partes del proceso.

Otro ejemplo de cambiar la definición del proceso en ejecución es cambiar los partners con los que el proceso se comunica. Es decir, si en un proceso BPEL un Web Service no está respondiendo dentro de un tiempo especificado, el proceso debería ser capaz de cambiar ese partner dinámicamente. Una completa clasificación en los cambios modulares puede se encuentra en [6].

* + 1. Maneras de implementar un motor workflow con aspectos
    2. Transformación del proceso

Se necesita una herramienta para unir la definición del proceso workflow y los aspectos, antes instanciar el proceso resultante en el motor. El beneficio de este proceso es que no es necesario hacer cambios al motor que ejecuta los procesos, lo que hace que las extensiones orientadas por aspectos sean independientes del motor donde se corren las instancias. Además, se tendrían dos versiones del proceso, (una antes y otra después del entretejido). La versión antes del entretejido es útil para entender el proceso abstrayendo los detalles técnicos. La versión después del entretejido es necesaria para entender correctamente y predecir cómo se va a ejecutar el proceso.

* + 1. Modificación del motor para verificación de aspectos

La otra opción es modificar el motor de workflow para que revise si tiene o no aspectos antes de ejecutar cada actividad. Con esta aproximación, el proceso no se modifica. La implementación de un motor *aspect-aware* es mucho más complicada que la implementación de una herramienta de transformación y lo hace dependiente de un motor en particular. Sin embargo, soporta la composición dinámica de aspectos y procesos, lo que permite tener un motor de workflow flexible y adaptable [14]. Hace posible que se puedan crear aspectos que modifican las instancias de procesos que se encuentran en ejecución. Estos aspectos podrían ser activados o desactivados sin la necesidad de modificar la definición del proceso y volverlo a ejecutar. Implementaciones actuales

* + 1. Aop4bpel

AO4BPEL es una tesis de doctorado desarrollada por Anis Charfi para la Universidad Técnica de Darmstadt [6, 11]. AO4BPEL es una extensión orientada por aspectos de BPEL, que soporta la modularidad de preocupaciones transversales y la adaptación dinámica de procesos BPEL.

En este motor, un aspecto es un documento XML que define un conjunto de pointcuts y de advices. Un pointcut es una expresión en XPath para seleccionar los join points. AO4BPEL soporta dos tipos de join points: join points en actividades, el cual permite seleccionar actividades específicas, y join points internos, el cual permite seleccionar puntos dentro de la ejecución de las actividades. Un pointcut se asocia con un advice, el cual es una acividad BPEL que define la lógica transversal. El tipo de advice (before, around, before soapmessageout) define el orden en que se ejecutan los advices. El lenguaje de advice permite el acceso al contexto de los join points, tales como el nombre de una actividad, sus entradas y salidas, los mensajes SOAP, etc. Adicionalmente, un aspecto AO4BPEL puede declarar partners, variables, fault handlers y compensation handlers. Actualmente existe una implementación de AO4BPEL, la cual está basada en IBM BPWS4J [11].

* 1. Aspect interference en workflows

En un ambiente de workflow, los problemas de interferencia que se dan en aplicaciones tradicionales, también ocurren. Por ejemplo, si se considera el caso donde es necesario conocer la cantidad de veces que una actividad es ejecutada y luego reportarlo. Si este nuevo requerimiento es dividido en dos aspectos: uno que incrementa el contador de ejecución de la actividad y otro aspecto que lo reporta a un supervisor. Si el primer aspecto es ejecutado después del segundo aspecto, siempre se recibirá información inconsistente.

El problema con la interferencia de aspectos es que los programadores desarrollan y prueban los aspectos de manera independiente. Debido a esto no se tiene en cuenta si otros aspectos van a ser ejecutados en el mismo punto. Tampoco se tiene en cuenta si se van a generar dependencias entre los aspectos. Una posible solución sería verificar si los aspectos interfieren unos con otros, pero hacer esto de manera manual tiene a facilitar cometer errores, aparte de lo tedioso que puede llegar a ser, ya que los conflictos no son obvios y mantener una gran cantidad de aspectos es muy difícil.

[1] Oasis. (2007, May). Web Services Business Process Execution Language Version 2.0. [En línea]. Disponible: <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel>

[2] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Pearson Education, 2005.

[3] B. Michelson. (2005, Sept.). Business Process Execution Language (BPEL) Primer. . [En línea]. Disponible: http://www.psgroup.com/detail.aspx?ID=630.

[4] Sanen, F. Truyen, E. Joosen, W. Jackson, A. Needos, A. Clarke, S. Loughran, N. Rashid, A. Classifying and Documenting Aspect Interactions. In: Workshop on Early Aspects at AOSD 2006.

[5] Durr, P. Bergmans, L. Aksit, M. Reasoning About Semantic Conflicts between Aspects. In: EIWAS 2005: 2nd European Interactive Workshop on Aspects in Software.

[6] Charfi, A. Aspect-Oriented Workflow Languages: AO4BPEL and Applications. Dissertation, Darmstadt D17, 2007 Accredited by the AOSD Europe Network of Excellence.

[7] R.E. Filman, T. Elrad, S. Clarke, M. Aksit. Aspect-oriented software development. Addison-wesley. 2005

[8] T. Piispanen. (2006, Oct.). BPEL & SOA, Reseatch Document. [En línea]. Disponible: <http://users.tkk.fi/~tpiispan/BPEL_SOA.pdf>

[9] IBM. WebSphere Process Server Version 6.0. <http://www-306.ibm.com/software/integration/wps>

[10] Oracle. BPEL Process Manager 10.1.2. http://www.oracle.com/technology/products/ias/bpel/index.html, August 2005.

[11] Charfi, A., Mezini, M. Aspect-Oriented Web Service Composition with AO4BPEL. In: Proceedings of the European Conference on Web Services (ECOWS)

[12] Charfi, A., Mezini, M. Aspect-Oriented Workflow Languages. In: Proceedings of the 14th International Conference on Cooperative Information Systems (CoopIS), November 2006

[13] Microsoft. BizTalk Server 2006. <http://www.microsoft.com/biztalk/default.mspx>.

[14] D. Townsend, R. Cairns, C. Schittko. (2003, Jul.). Using BPEL4WS- What IT Managers need to know. [En línea]. Disponible: http://www.openstorm.com/articles/BPEL-WSJ-Article.pdf

[15] T. Piiespanen. (2006, Oct.). BPEL- SOA, Research Document. [En línea]. Disponible: <http://users.tkk.fi/~tpiispan/BPEL_SOA.pdf>

[16] ActiveBpel. (2007). ActiveBpel Open Source Engine. [En línea]. Disponible: <http://www.active-endpoints.com>.

[17] Apache. (2008) Apache ODE [En línea]. Disponible: <http://ode.apache.org/>