Reingeniería y ampliación del generador dinámico de invariantes potenciales para composiciones de servicios web en WS-BPEL Takuan

Alejandro Álvarez Ayllón

21 de diciembre de 2009

Página dejada intencionadamente en blanco.

Escuela Superior de Ingeniería

Ingeniero en Informática

Reingeniería y ampliación del generador dinámico de invariantes potenciales para composiciones de servicios web en WS-BPEL Takuan

Departamento: Lenguajes y Sistemas Informáticos

Director del proyecto: Manuel Palomo Duarte

Autor del proyecto: Alejandro Álvarez Ayllón

Cádiz, 21 de diciembre de 2009

Licencia

Notación

En el presente documento se sigue la siguiente notación: Los términos extranjeros (p.ej, inglés) aparecerán *en cursiva*; los nombres de ficheros tendrán una `fuente monoespaciada`; los elementos de un documento XML estarán contenidos <entre símbolos de mayor y menor>.

Extractos de código XML aparecerán

```
1 Dentro de un recuadro como este
```

NOTA: Para asegurar la compatibilidad en cualquier codificación, dentro de los bloques de código se han omitido los acentos.

Mientras que extractos de documentos en texto plano - como pueden ser las trazas - aparecerán

```
En bloques monoespaciados.
```

Por último, cuando se indique un comando a ejecutar en línea de consola, se usará una fuente monoespaciada, y el comando precedido por el símbolo del dólar.

```
$ comando parámetros >> redirección
```

Índice general

Capítulo 1

Introducción

1.1. Sobre los SOA

Los SOA (*Service Oriented Architecture*, Arquitectura Orientada a Servicios) es un paradigma para organizar y utilizar elementos distribuidos que pueden estar bajo el control de diferentes dueños [OAS06].

Los servicios web (WS, de *Web Services*) son una forma frecuente de implementar este paradigma, empleando comunicaciones HTTP y serialización XML, por lo que es independiente del sistema operativo, plataforma hardware y lenguaje de programación.

Por último, WS-BPEL[1] es un estándar definido por OASIS que permite especificar interacciones entre servicios web, siendo estos sus únicos métodos de entrada y/o salida.

1.2. Objetivos

1.2.1. Funciones

Los objetivos de este PFC se pueden agrupar en tres líneas:

1. Análisis de las limitaciones del sistema Takuan [PD08b], desarrollado por el grupo de investigación "Mejora del Proceso Software y Métodos Formales" del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz.

   Este sistema permite generar dinámicamente invariantes observados en una composición de servicios web con WS-BPEL (el lenguaje estandarizado por OASIS para orquestar servicios web). Sin embargo, debido a las instrucciones específicas que incluye el lenguaje WS-BPEL, así como a la arquitectura interna del sistema, actualmente presenta ciertas limitaciones que sería interesante analizar.

2. Propuesta de transformación de flujos de datos y de cambios en la arquitectura interna para superar las limitaciones detectadas. La propuesta irá acompañada de desarrollos con ejemplos.

[1] Las versiones 1.0 y 1.1 se llamaron BPEL4WS. Se renombró en la versión 2.0 para respetar la convención de nombres de las tecnologías relacionadas con los servicios web.

3. Implantación de un traductor que incluya las transformaciones anteriores, aplicación a diversos ejemplos y evaluación de los resultados.

1.2.2. Elementos innovadores a desarrollar

El lenguaje WS-BPEL, eje vertebrador de este trabajo, es el resultado del esfuerzo de las principales empresas TIC [2] a nivel mundial (Microsoft, IBM, SAP, Oracle, etc.) por estandarizar la composición de servicios web [Com07]. Actualmente WS-BPEL está soportado por la mayoría de herramientas líderes del mercado SOA [Dom07]. De este modo, se abre la puerta a la adopción masiva de dicho lenguaje de programación a gran escala (*programming in the large*) en el mercado de las TIC en los próximos años.

Como consecuencia, se presenta la oportunidad de investigar los elementos que permitan asegurar la calidad de dichas composiciones. En concreto nos centraremos en la generación de invariantes con Takuan, una herramienta presentada en congresos internacionales de reconocido prestigio [PD08b, PD08a, Pal09a].

1.2.3. Aplicabilidad y alcance

El trabajo ha sido desarrollado como apoyo dentro del proyecto SOAQSIM (TIN-2007-67843-C06-04) financiado por el Programa Nacional de I+D+i del Ministerio de Educación y Ciencia y fondos FEDER, en el que participa el grupo de investigación "Mejora del Proceso Software y Métodos Formales" del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz.

Asimismo, para la realización de parte de este trabajo se me concedió una beca de colaboración del *Ministerio de Educación, Política Social y Deporte*.

1.3. Definiciones, acrónimos y abreviaturas

A continuación se describen algunos términos empleados para facilitar la lectura del documento:

Ant Herramienta orientada a la automatización de construcción de software. Mantenida por la Fundación Apache [Apa].

BPEL Ver WS-BPEL.

BPR Extensión del fichero JAR que contiene todos los ficheros necesarios para realizar el despliegue de un proceso de negocio.

Desplegar En el ámbito de los SOA, *desplegar* un proceso se refiere a poner en funcionamiento un servicio web (escuchando en un determinado puerto).

GPL *GNU General Public License*. Es actualmente la licencia de software libre más extendida.

GUI *Graphical User Interface*, Interfaz Gráfica de Usuario.

HTTP *Hypertext Transfer Protocol* [IET], protocolo de comunicación sobre Internet. Diseñado para intercambiar principalmente documentos enlazados entre ellos. Llevó a la aparición de la *Worl Wide Web*.

[2] Tecnologías de la Información y la Comunicación

IDE *Integrated Development Environment*, Entorno Integrado de desarrollo. Software que proporciona un marco de trabajo para uno o más lenguajes de programación. Pueden verse algunos IDEs para WS-BPEL en el apéndice C.

Instrumentalizar e instrumentar Preparar el código o el binario de un programa para extraer la información deseada durante su ejecución (del inglés *instrument*). Según la RAE ambos términos son equivalentes en nuestro contexto y se usan por igual en el presente documento.

Invariante En este documento se considera "invariantes" a aquellas propiedades que se mantienen constante - siempre o para un conjunto de prueba - en un punto dado de un programa. Se ha adoptado esta definición al ser la empleada por trabajos relacionados [Pal08, Ern01].

JAR *Java ARchiver*. Es un formato de archivo que contiene un programa o librería Java empaquetado.

Mockup Un *mockup* es un servicio sustituto de un servicio real en una simulación, que implementa un comportamiento predefinido, y que suele emplearse en pruebas para reemplazar a un servicio costoso, al que no se puede acceder, o también para comprobar si la salida de un programa para un caso de prueba es la esperada. No tienen lógica interna, limitándose a responder con mensajes predefinidos, o "fallando" si así se especifica.

PDD *Process Deploy Descriptor*. Fichero empleado en ActiveBPEL que contiene la información necesaria para desplegar un proceso WS-BPEL.

Proceso síncrono Proceso en el que el proceso cliente queda a la espera de la respuesta del proceso servidor.

Proceso asíncrono Una vez realizada la solicitud, será el proceso servidor quién responderá cuando esté listo.

Punto de programa Instrucción sequence o flow que se instrumenta para posteriormente generar invariantes justo antes y después de su ejecución.

SOAP Es un protocolo ligero orientado al intercambio de información estructurada en un entorno descentralizado y distribuido. Se construye sobre XML y es la base de los SOA [W3C07a].

UDDI *Universal Description, Discovery and Integration* [OAS05]. Registro basado en XML empleado por las empresas para listar sus servicios en Internet. Se diseñó para ser interrogado por mensajes SOAP para obtener documentos WSDL.

URI *Uniform Resource Identifier*, cadena de caracteres que identifican de forma unívoca un recurso en Internet.

URL *Uniform Resource Locator*, subconjunto de URI que especifica dónde se encuentra un recurso, y cómo acceder a él.

Web Services Los servicios web son sistemas software diseñados para soportar interacciones máquina-a-máquina sobre una red. Tiene una interfaz descrita en un formato procesable (concretamente, WSDL). Otros sistemas pueden interactuar con el servicio web de una forma que se ajuste a esta descripción empleando mensajes SOAP. Normalmente estos mensajes son transmitidos a través de HTTP con una serialización XML junto a otros estándares web [Haa04].

WS-BPEL *Web Services Business Process Execution Language* [Bar07]. Es un lenguaje diseñado para especificar procesos de negocios basados en Servicios Web (*Web Services*). Más información en el apéndice B. Tambiés es referido frecuentemente como BPEL sólamente. Ambas nomenclaturas son equivalentes y se emplean por igual.

WSDL *Web Services Description Language*, lenguaje basado en XML usado para describir servicios web [W3C01]. Más información en el apéndice A.

XML *eXtensible Markup Language*, especificación para crear lenguajes de marcado. Su principal objetivo es facilitar el intercambio de información estructurada, especialmente vía Internet.

XML Schema Lenguaje empleado para definir esquemas XML, esto es, la estructura que debe cumplir un documento XML para ser considerado válido respecto a ese mismo esquema. Este mismo lenguaje es una extensión de XML [W3C09].

XPath Lenguaje para seleccionar nodos de un documento XML. Puede emplearse también para procesar valores contenidos en dicho documento [W3C07c].

XSLT *XSL Transformations* [W3C99]. Es un lenguaje basado en XML para transformar documentos XML en otros documentos XML, o, incluso, texto plano.

1.3.1. Estructura del documento

El resto del documento se divide en cuatro partes diferenciadas:

1. En la primera parte se introduce la plataforma Takuan, y documenta su diseño y funcionamiento.
2. En la segunda parte se describe todo el trabajo realizado sobre Takuan, desde la obtención de ejemplos hasta la obtención de información sobre cobertura, pasando por la corrección de errores. También se exponen las vías de trabajo futuro.
3. En la tercera parte se describe paso a paso cómo usar Takuan, desde su obtención e instalación hasta la interpretación de los resultados.
4. En la cuarta y última parte se incluyen cuatro apéndices, dos para facilitar la comprensión de las tecnologías WS-BPEL y WSDL - imprescindibles para comprender el funcionamiento de Takuan - y otros dos sobre editores visuales y el servidor Neptuno, que no están relacionados diréctamente con la realización del proyecto, pero que se han hecho de forma paralela.

Por último, se incluye toda la bibliografía empleada.

Parte I

Takuan

Figura 2.1: Logotipo de Takuan

Capítulo 2

Takuan en la prueba de composiciones WS-BPEL

El nombre del sistema es en realidad un juego de palabras entre los dos posibles significados del término:

1. El generador dinámico de invariantes empleado se llama Daikon, que es un término japonés que define un tipo de rábano. A su vez, Takuan es un plato típico basado en daikon (rábano) marinado. Como Takuan se basa en el sistema Daikon, de ahí el término.

2. Takuan es también el nombre de un importante monje budista japonés: Takuan Soho.

Esta dualidad es reflejada en el logotipo del programa (figura 2.1).

2.1. Prueba de caja blanca de software

Existen, principalmente, dos maneras de probar un producto software [Pal08, Ber05, Mye04]:

Pruebas *black-box* Se trabaja únicamente sobre las entradas y salidas de un programa, sin entrar en detalles internos.

Pruebas *white-box* Se tiene en cuenta la lógica interna.

El segundo método requiere, lógicamente, acceso al código, pero sus resultados son mucho más afinados.

Una de las técnicas más empleadas hasta ahora para probar la validez de algoritmos han sido la inferencia manual de invariantes. Sin embargo, su extracción puede automatizarse. De hecho, la generación automática de invariantes ha demostrado ser una técnica eficaz para ayudar en la prueba de caja blanca y la mejora de programas escritos en lenguajes de programación estructurados y orientados a objetos [Ern01].

Estos invariantes pueden aplicarse para [Ern01, Ern07]:

Depurar el código Ya que un invariante inesperado puede sacar a relucir un fallo en el código que, de otra forma, podría haberse pasado.

Actualizar el software sin alterar las invariantes que deban conservarse. Tras comprobar qué invariantes deben permanecer inalterados en la siguiente versión, puede desarrollarse ésta y después comparar los nuevos invariantes con los esperados.

Documentar el código añadiendo aquellos invariantes que puedan ayudar a comprenderlo mejor.

Verificar si la composición se ajusta a las especificaciones

Validar el conjunto de casos de prueba , determinando si es lo suficientemente amplio y/o variado, ya que un falso invariante puede indicar un conjunto de casos de prueba insuficiente.

La extracción automática de invariantes, a su vez, puede realizarse de dos formas diferentes:

Estática [Bjø97, Col03], sin ejecutar realmente el programa. Se realiza un análisis del código - en especial flujos de datos y de control - para extraer los invariantes. Los invariantes así obtenidos siempre son correctos, pero las limitaciones de la máquina formal, en especial con lenguajes como WS-BPEL, restringe la cantidad de datos que se pueden obtener.

Dinámica [Ern01]. En este caso los invariantes se infieren a partir de logs de ejecución del programa objetivo. Es decir, no se analiza el código en sí mismo, sino las trazas de ejecución. Por ello, si el conjunto de casos de prueba no es lo bastante amplio, podrán aparecer falsos invariantes. En ese caso necesitaríamos un conjunto más completo de casos de prueba para eliminarlos de la salida.

2.2. Generación dinámica de invariantes para WS-BPEL

Hasta ahora la mayor parte de las aproximaciones de caja blanca para WS-BPEL se han basado en modelos de simulación orientados a pruebas [Buc07], incluso a veces traduciendo el código WS-BPEL a un lenguaje secundario extrayendo su lógica interna.

Simular un motor WS-BPEL no es algo trivial, ya que hay una gran cantidad de características complejas que deben ser implementadas, como el manejo de fallos, de excepciones, concurrencia o eventos. Si una de estas características no se implementara correctamente, no se podrían probar las composiciones [PD08b].

Es por este motivo por el que la aproximación de Takuan es ejecutar las composiciones sobre un motor WS-BPEL real, generando los invariantes a partir de las trazas de ejecución. Generalmente, cuantas más trazas se proporcionen, mejores resultados se obtendrán, aunque también dependerán de que el conjunto de casos de prueba sea lo suficientemente completo como para cubrir la composición - cobertura de ramas y de instrucciones, ver capítulo 8.

En el caso de que el conjunto de caso de pruebas sea insuficiente, posiblemente se obtendrán falsos invariantes. Al estudiar estos invariantes se podrían descubrir deficiencias en el conjunto de casos de prueba. Y, una vez mejorado el conjunto, estos invariantes falsos desaparecerían.

Otra ventaja de este sistema es que no se usan lenguajes intermediarios, lo que evita errores derivados de una mala traducción, o de una simulación incorrecta.

Por último, esta técnica permite realizar pruebas aún cuando no todos los servicios externos estén disponibles, sustituyéndolos por otros con un comportamiento predefinido (*mockups*). Cómo se hace esto se verá en el capítulo 3, que trata sobre la arquitectura de Takuan.

Capítulo 3

Arquitectura

El esquema de la arquitectura de Takuan [Gar08a] puede verse en la figura 3.1. Es un sistema basado en tuberías dividido en tres bloques básicos: instrumentación, ejecución y análisis [Gar08b].

3.1. Fase de instrumentación

Takuan emplea el software Daikon [Ern07] como detector de invariantes, quien infiere invariantes a partir de las trazas de ejecución de un programa, no a partir de su código. Esto supone una gran ventaja, ya que hace que este motor de inferencia sea independiente del lenguaje a tratar, pero también hace necesario obtener de alguna forma las trazas y convertirlas al formato de entrada aceptado por Daikon.

3.1.1. Descripción

Una forma de obtenerlas es a través de un proceso llamado "instrumentación"; esto es, modificar el programa, ya sea su código o su forma compilada, de forma que se generen unos ficheros que Daikon necesita (en concreto ".dtrace", que contiene el historial de la ejecución).

Para Daikon la instrumentación es llevada a cabo por "front-ends", existiendo ya implementaciones para instrumentar programas en Java o en C/C++ en su forma compilada, Eiffel, Lisp, ... Virtualmente, cualquier lenguaje puede ser instrumentado. Básicamente añade instrucciones que imprimen los valores da las variables del programa en un punto determinado.

Es aquí donde entra la fase de instrumentación en Takuan. Básicamente es un proceso que tiene como entrada el proceso a inspeccionar - fichero BPEL, documentos WSDL, declaraciones XSD, ... - y produce como salida:

procesoInspeccionado.bpel Un equivalente al proceso original, pero incluye una serie de actividades adicionales que permitirán ir generando la traza.

catalog.xml Un fichero que contiene todos las dependencias del proceso BPEL.

proceso.pdd Fichero con la configuración necesaria para el despliegue del proceso en el servidor ActiveBPEL.

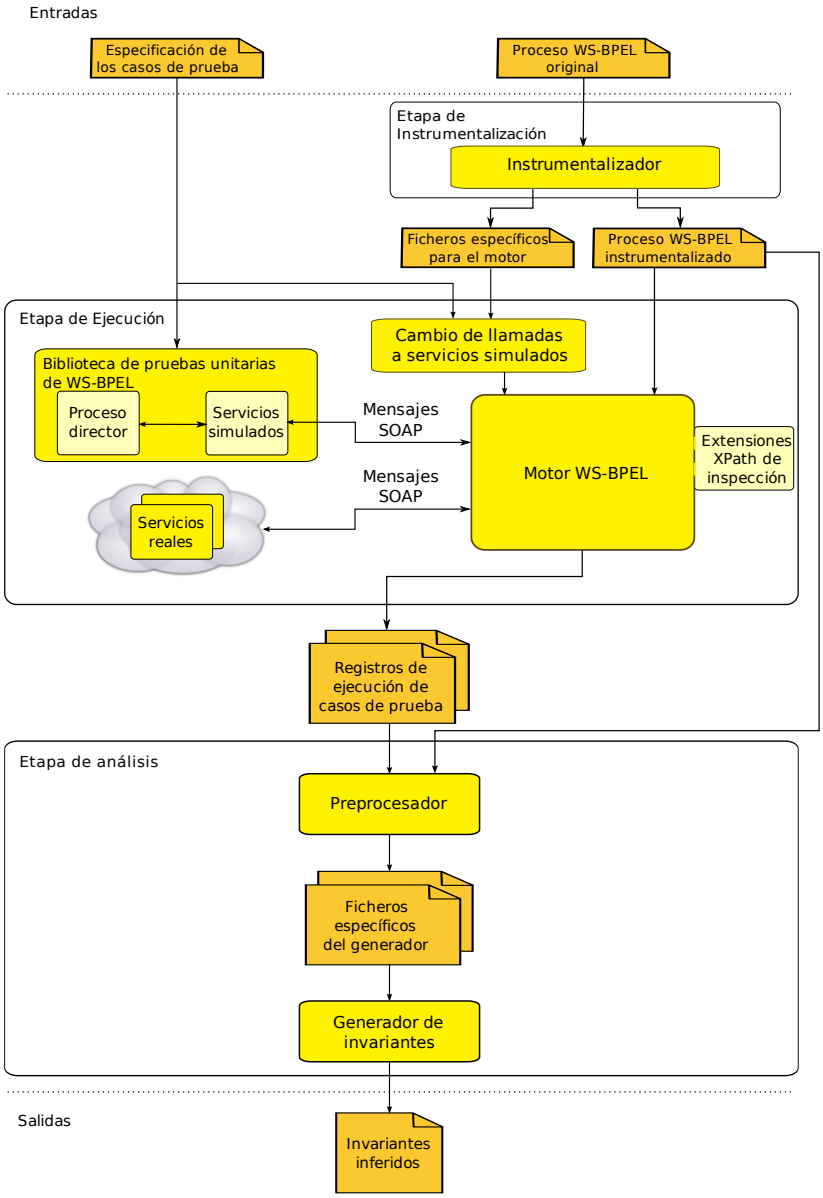

Figura 3.1: Arquitectura general del framework Takuan.

tiposReunidos.xml Agrupa las declaraciones de tipos y propiedades de todos los XML Schema y WSDL.

procesoInspeccionado.decls Contiene información sobre la comparabilidad de las variables inspeccionadas (esto se explicará en el capítulo ***).

Todos estos ficheros, junto a los documentos WSDL, XML Schema originales y conjunto de casos de prueba son empaquetados en un fichero ".bpr" que contiene toda la información y estructura necesarias para desplegar el proceso.

Para más información sobre cómo definir un conjunto de casos de prueba y mockups, puede consultar el manual de usuario (parte III).

3.1.2. Funcionamiento

Dado que la salida de esta fase es la entrada de la siguiente (ejecución), es necesario primero conocer qué tecnologías se van a emplear en ella, para saber qué necesitamos como entrada.

Pero para saber las tecnologías concretas, debemos definir qué necesitamos [Pal08]:

- Un motor WS-BPEL que nos permita ejecutar el proceso, sustituyendo las llamadas a servicios reales por otras declaradas por el usuario. Esta sustitución podemos desear hacerla por tres motivos:
  1. Los servicios reales pueden no estar disponibles, o no queremos interferir con ellos
  2. Los servicios reales son de pago, bloquean recursos, etc. - algo bastante común.
  3. Queremos establecer de antemano cuáles van a ser las respuestas o peticiones de los servicios externos, de forma que podamos controlar en todo momento el entorno de prueba.
- Una librería de prueba unitaria que desplegará el servicio, y que actuará como cliente - invocando el servicio a probar - y como servidor para los mockups. Esta librería será más complejas que las de otros lenguajes, ya que debe actuar:
  1. Como "director", preparando y monitorizando la ejecución del conjunto de casos de pruebas.
  2. Como un servidor de "mockups", manejando las peticiones entrantes y actuando como el servicio externo requerido por el proceso WS-BPEL.

En función de estos requerimientos, se escogió ActiveBPEL [Act08] como motor de ejecución, y BPELUnit [May06b] como librería de pruebas unitarias. Ambas plataformas distribuidas bajo licencias libres, lo que permite realizar modificaciones en caso de ser necesario.

Una vez escogidos los productos concretos, volvemos al problema de cómo conseguir las trazas. Aunque ActiveBPEL puede generar trazas sobre los flujos de control - condicionales, bucles, ...- , no permite detallar los valores de las

variables empleadas. Es necesario ampliar de alguna forma el funcionamiento habitual del motor para poder examinar los valores de las variables - de aquí el interés por usar software libre.

Este problema ha sido solucionado creando funciones que extienden el lenguaje XPath [W3C07c], definidas dentro de un espacio de nombres propio, que permiten examinar los valores de las variables. Esto ha sido posible gracias a que ActiveBPEL permite añadir extensiones a las funciones estándares y a las suyas propias.

Sin embargo, una limitación de WS-BPEL es que no pueden llamarse funciones XPath directamente, por lo que es necesario emplear una estructura de copia y asignación a una variable "dummy" cuyo valor no se empleará. También será necesario deshabilitar los errores provocados por usar variables no inicializadas [Pal08].

El último paso en esta fase es la creación de ficheros específicos de ActiveBPEL necesarios para el posterior despliegue y ejecución del proceso instrumentado.

3.1.3. Implementación

Como tanto el fichero WS-BPEL, como los documentos de apoyo - WSDL, XML Schema - son todos documentos XML, XSLT [W3C99] es una buena opción para realizar las transformaciones. Con vistas a automatizar el proceso, se definen también clases Java que podrán ser ejecutadas desde scripts Ant [Apa].

El proceso se entiende mejor a través de la figura 3.2.

- La clase *IntrumentBPELProcessTask* extiende la clase *org.apache.tools.ant.Task*, lo que le permite ser llamada desde scripts Ant. Su único cometido es llamar secuencialmente a los métodos que proporciona *BPELProcessInstrumenter*.

- La clase BPELProcessInstrumenter analiza los ficheros originales y genera el fichero de despliegue (proceso.pdd) y el catálogo, un fichero donde se especifican los documentos de los que depende el proceso.

- Al fichero BPEL original se le aplica la hoja de estilos "hojaTipos.xslt", reuniendo en un único fichero todas las declaraciones de tipos, propiedades, etc. (Nota: También se procesarán los documentos WSDL y XSD asociados).

- Al fichero BPEL original se le aplica la hoja de estilos "hojaInspeccionar.xslt" (que depende de "hojaComparabilidad"), generando las instrucciones precisas para poder trazar el contenido de las variables. El resultado se almacenará en "procesoInspeccionado.bpel". Esta hoja:

  1. Añade al comienzo y final de cada bloque (secuencias, flujos... ver apéndice B) instrucciones para obtener el valor antes y después - precondiciones y postcondiciones.
  2. Las asignaciones originales son "envueltas" en llamadas XPath con la forma uca:ambito/uca:ambitoPropiedad, de forma que pueda obtenerse información sobre comparabilidad.

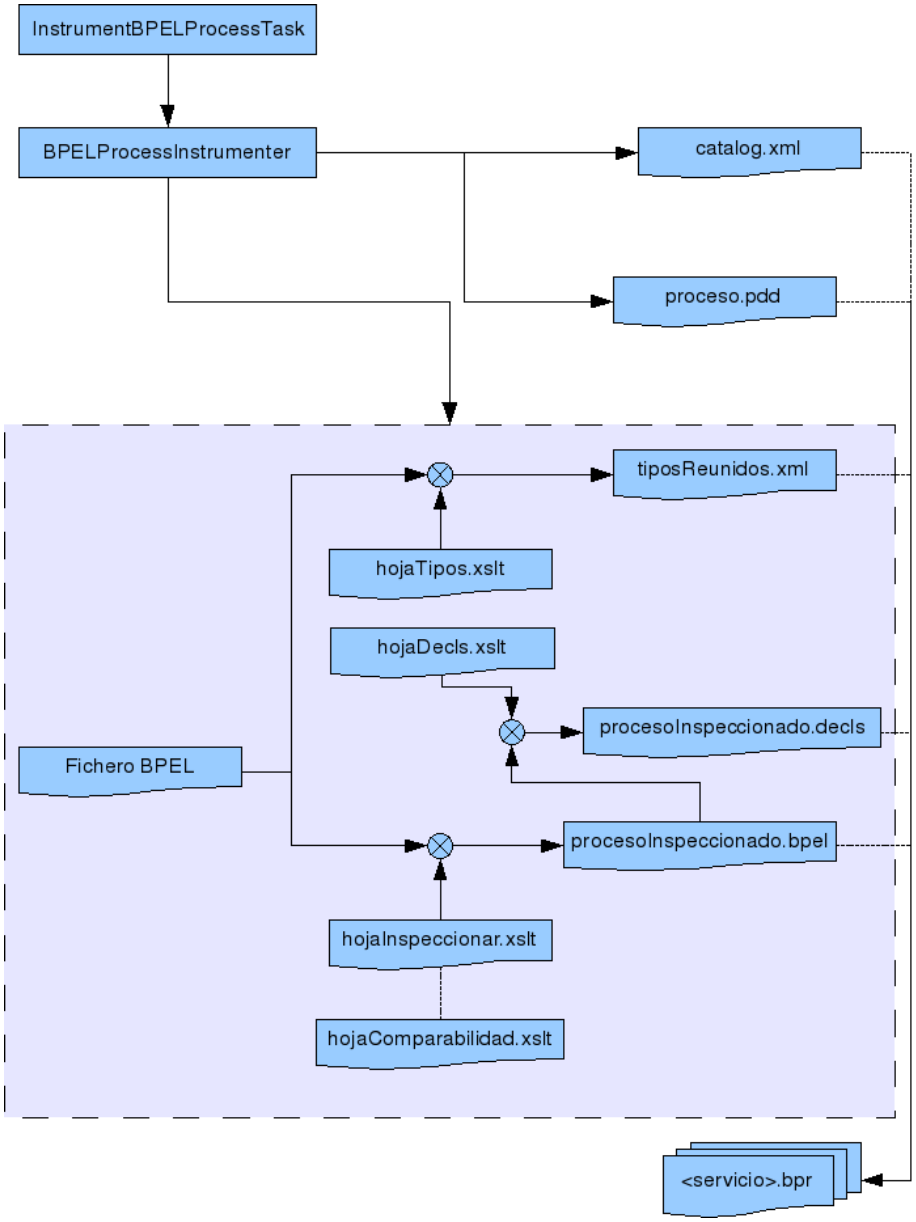

Figura 3.2: Esquema de la fase de instrumentación

3. Se envuelve en un sequence para asegurar ejecución secuencial en caso de estar dentro de un elemento paralelo como flow.

- Al proceso ya inspeccionado se le aplica la hoja de estilos "hojaDecls.xslt", obteniendo un fichero, con un formato reconocido por Daikon, que incluye información sobre los ámbitos y comparabilidades de las variables.
- Por último, se empaquetan todos los ficheros resultantes, junto con los documentos originales, en un fichero BPR, que no es más que un paquete en formato ZIP. Este fichero es la salida del proceso, y será la entrada de la siguiente fase.

Los detalles internos de la implementación de las clases Java - y de sus clases de apoyo - no se cubrirán aquí. Su documentación se encuentra en el CD adjunto.

3.1.4. Resumen

Entradas Fichero BPEL original, hojas WSDL, declaraciones XSLT.

Salidas Ficheros específicos del motor WS-BPEL para el despliegue, fichero BPEL modificado.

Fase previa Ninguna (Entrada suministrada por el usuario).

Fase siguiente Fase de ejecución.

Tecnologías empleadas Java, hojas de transformación (XSLT).

3.1.5. Ejemplo

A continuación se mostrarán ejemplos de determinadas partes de un proceso antes y después de instrumentalizarlo.

Declaración de variables

Listing 3.1: Código de declaración de variables antes de la instrumentalización

```
1 <variables>
2   <variable messageType="tns:sellerInfoMessage" name="
        sellerInfo"/>
3   <variable messageType="tns:negotiationMessage" name="
        negotiationOutcome"/>
4   <variable messageType="tns:buyerInfoMessage" name="
        buyerInfo"/>
5 </variables>
```

Listing 3.2: Código de la declaración de variables después de la instrumentalización

```
1 <variables>
2   <variable messageType="tns:sellerInfoMessage" name="
        sellerInfo"/>
```

```
3   <variable messageType="tns:negotiationMessage" name="
        negotiationOutcome"/>
4   <variable messageType="tns:buyerInfoMessage" name="
        buyerInfo"/>
5   <variable messageType="tns:sellerInfoMessage" name="
        dummy_sellerInfo"/>
6   <variable messageType="tns:negotiationMessage" name="
        dummy_negotiationOutcome"/>
7   <variable messageType="tns:buyerInfoMessage" name="
        dummy_buyerInfo"/>
8 </variables>
```

Asignaciones

Listing 3.3: Código de una asignación antes de instrumentalizar

```
1 <assign>
2   <copy>
3     <from variable="shipRequest"
4           property="props:shipOrderID" />
5     <to variable="shipNotice"
6           property="props:shipOrderID" />
7   </copy>
8 </assign>
```

Listing 3.4: Código de una asignación después de instrumentalizar

```
1  <!-- Codigo de inspeccion (al principio y final del
       bloque) -->
2  <bpel:copy ignoreMissingFromData="yes">
3    <bpel:from>uca:inspeccionarPropiedad('shipRequest', '
         props:shipOrderID')</bpel:from>
4    <bpel:to>$dummy_shipRequest_shipOrderID</bpel:to>
5  </bpel:copy>
6
7  <!-- Codigo de la asignacion -->
8  <assign>
9    <copy>
10     <bpel:from xmlns:vprop="http://docs.oasis-open.org/
           wsbpel/2.0/varprop"
11                 xmlns:wsdl="http://schemas.xmlsoap.org/
                       wsdl/">
12       uca:ambitoPropiedad('shipRequest','
             props:shipOrderID')</bpel:from>
13     <to variable="shipNotice" property="props:shipOrderID
           " />
14   </copy>
15 </assign>
```

Comparaciones

Listing 3.5: Código de una condición antes de instrumentalizar

```
1 <condition>
2   $itemsShipped &lt; bpel:getVariableProperty('
        shipRequest', 'props:itemsTotal')
3 </condition>
```

Listing 3.6: Código de una condición después de instrumentalizar

```
1 <condition>
2   uca:ambito(0, '(uca:imprimirRutas(1, @0@$
        itemsShipped@0@) &lt; uca:ambitoPropiedad(
        @0@shipRequest@0@, @0@props:itemsTotal@0@))')
3 </condition>
```

El código "@0@" es una nomenclatura empleada para poder distinguir los niveles de anidamientos que puedan existir. Durante su interpretación, se sustituirán por comillas simples.

3.2. Fase de ejecución

3.2.1. Descripción

Una vez generado el código WS-BPEL instrumentalizado, se ejecutará sustituyendo los servicios externos con los servicios simulados.

Por cada caso de prueba se generará un fichero con la traza de la ejecución. Concretamente los pasos son:

1. Desplegar la composición WS-BPEL (la modificada) en el motor WS-BPEL. En este caso, se envía un archivo comprimido (.bpr) que contiene la lógica y algunos archivos específicos.
2. Se inicia el servidor que simula las respuestas externas.
3. Por cada caso de prueba
   a) Se configura el servidor "mockup" con las respuestas predefinidas, ó incluso su ausencia, si queremos simular un servicio inaccesible.
   b) Invocar el proceso que está siendo probado
   c) Obtener los resultados
4. Detener el proceso WS-BPEL

3.2.2. Implementación

Es en esta fase cuando BPELUnit - concretamente una versión modificada - adquiere el control del proceso. Recibe como entrada un fichero .bpr con todos los ficheros necesarios, incluyendo uno con extensión .bpts que contiene la especificación del conjunto de casos de prueba, los mockups que reemplazarán a los servicios reales y el comportamiento del cliente simulado.

A partir de los mockups especificados, BPELUnit modifica el fichero BPEL asegurándose de que las llamadas realmente se realicen a los servicios simulados, tras lo cual indicará a ActiveBPEL que despliegue el proceso.

Una vez que el proceso esté listo, BPELUnit se conectará al mismo actuando como cliente, y ejecutando la operación indicada en el caso de prueba en el puerto correspondiente. En el caso de que el proceso inspeccionado realice alguna llamada a un servicio externo, será BPELUnit quien reciba la petición, y responderá en función a lo especificado en el fichero bpts. Este proceso se repetirá por cada caso de prueba.

El servidor ActiveBPEL tendrá instalado por su parte una extensión de XPath que ejecutará las siguientes funciones relacionadas con la instrumentalización.

uca:inspeccionar Inspecciona el valor de una variable para incluirla en la salida.

uca:ambito Empleada para obtener el ámbito y la comparabilidad de una variable (más información en el capítulo III).

uca:imprimirRutas Imprime la ruta de una variable (p.ej `$variable/nodo[1]/hoja[0]`)

uca:inspeccionarPropiedad Igual que uca:inspeccionar, pero para propiedades de variables WS-BPEL.

uca:ambitoPropiedad Igual que uca:ambito, pero para propiedades

Estas extensiones imprimen la información requerida en los logs de ejecución que ActiveBPEL genera por cada proceso. Estos logs serán la entrada de la siguiente fase.

3.2.3. Resumen

Entradas Ficheros específicos del motor WS-BPEL para el despliegue, fichero BPEL modificado, conjunto de casos de prueba (todo empaquetado junto)

Salidas Trazas de ejecución

Fase previa Fase de instrumentación

Fase siguiente Fase de análisis

Tecnologías empleadas ActiveBPEL, BPELUnit, extensiones XPath

3.2.4. Ejemplo

Se muestra un extracto de una traza de ejecución:

Listing 3.7: Extracto de traza de ejecución

```
1 [1][2009-09-23 09:44:48.938] : Executing [/process/
    sequence/if/else/sequence/while]
2 [1][2009-09-23 09:44:48.938] : ---- INICIO AMBITO ---- []
3 [1][2009-09-23 09:44:48.944] : Ruta accedida: $
    itemsShipped []
```

```
4  [1][2009-09-23 09:44:48.946] : Ruta accedida: $
       shipRequest~props:itemsTotal []
5  [1][2009-09-23 09:44:48.947] : ----- FIN AMBITO ----- []
6  [1][2009-09-23 09:44:48.947]     While Condition for  is
       true :  [/process/sequence/if/else/sequence/while]
7  [1][2009-09-23 09:44:48.947] : Executing [/process/
       sequence/if/else/sequence/while/sequence]
8  [1][2009-09-23 09:44:48.947] : Executing [/process/
       sequence/if/else/sequence/while/sequence/assign]
9  [1][2009-09-23 09:44:48.948] : Executing [/process/
       sequence/if/else/sequence/while/sequence/assign/copy
       [0]]
10 [1][2009-09-23 09:44:48.951] : INSPECTION($shipRequest.
       shipOrder/ship:shipOrder[1]/
       ship:ShipOrderRequestHeader[1]/ship:shipOrderID[1]) =
       555 []
11 [1][2009-09-23 09:44:48.952] : INSPECTION($shipRequest.
       shipOrder/ship:shipOrder[1]/
       ship:ShipOrderRequestHeader[1]/ship:shipItemsTotal[1])
        = 1 []
12 [1][2009-09-23 09:44:48.958] :
```

En el extracto anterior se pueden ver las salidas generadas para las funciones de inspección y las de ámbito. Si nos fijamos en el texto comprendido entre INICIO AMBITO y FIN AMBITO podemos deducir que $itemsShipped y la propiedad itemsTotal de $shipRequest son comparables. Esta información resultará de gran utilidad para Daikon.

3.3. Fase de análisis

3.3.1. Descripción

En esta fase se dispone de varios ficheros con las trazas de cada caso de prueba en el formato específico de ActiveBPEL, por lo que lo primero es convertir estas trazas al formato que Daikon acepta como entrada [Ern01].

La transformación de un formato al otro no es trivial debido a la naturaleza de la estructura de los tipos de WS-BPEL y las limitaciones de Takuan. Especialmente delicado es el tratamiento de tipos no escalares, dado que Daikon razona internamente con tipos de datos Java, por lo que no es capaz de manejar estructuras más complejas que vectores unidimensionales.

Takuan contempla dos técnicas para abordar este problema [PD08a] que se explicarán sobre el ejemplo 3.8.

Listing 3.8: Ejemplo del contenido de una variable compleja

```
1 <MetaSearchProcessResponse>
2   <noResult>4</noResult>
3   <noFromGoogle>1</noFromGoogle>
4   <noFromMSN>3</noFromMSN>
5   <result>
6     <url>http://url1google</url>
```

```
7       <title>Title1google</title>
8       <snippet>Snippet1google</snippet>
9       <from>Google</from>
10    </result>
11    <!-- (three results from MSN) -->
12  </MetaSearchProcessResponse>
```

Matrix slicing

Esta aproximación - basada en la tomada por Kvasir, el frontend de C++ para Daikon - consiste en dividir un vector de N dimensiones iterativamente en vectores de N-1 dimensiones, hasta que sólo queden vectores unidimensionales.

En el ejemplo 3.8 tenemos cuatro variables de dos dimensiones cada una, agrupada por resultados: result[].url[], result[].tile[], result[].snippet[] y result[].from[]. Podrían ser separadas de forma individual, obteniendo 16 variables - result[0].url[], result[1].url[], result[2].url[], ...

- Es el sistema más natural para obtener invariantes sobre elementos individuales de estructuras multidimensionales, ya que cada uno de ellos estaría contenido en una variable independiente.
- Todas las variables se corresponden a un conjunto contiguo de elementos del árbol XML original. De esta forma pueden filtrarse invariantes redundantes sobre la longitud de un vector que ya están recogidos en la definición XML Schema.
- Sin embargo, el número de variables crece muy rápidamente, empeorando con estructuras de más dimensiones y más elementos. Esto podría aumentar los requerimientos de memoria y tiempo de ejecución considerablemente, ya que implica trazas mayores y más combinaciones posibles para comprobar. Se hace necesario seleccionar de forma cuidadosa las variables y puntos a analizar.
- Daikon no puede comprobar cualquier posible combinación de variables por motivos de complejidad y eficiencia, limitándose a invariantes que relacionan 1, 2 o 3 variables como máximo. Esto significa que Daikon no podrá generar invariantes relacionados con estructuras multidimensionales divididas en más de tres variables, aunque algunos tipos pueden obtenerse usando variables derivadas, como result.length.

Matrix flattening

WS-BPEL emplea el estándar XPath como su lenguaje por defecto para describir asignaciones, condiciones booleanas, etcétera. La forma en la que XPath permite acceder a un elemento B bajo uno A es de la forma A/B.

Esta forma de examinar elementos anidados puede usarse como base para reducir las dimensiones de una matriz: aplanándola en un único vector unidimensional siguiendo un orden predefinido de navegación, concretamente el original de un documento XML. Esto es, a[1]/b[2] es anterior a a[2]/b[1].

Volviendo al ejemplo 3.8, las cuatro variables bidimensionales pueden ser aplanadas en cuatro variables unidimensionales, cada una con cuatro elementos.

Método	Puntos de programa	Variables	Tiempo (mm:ss)
Slicing	64 (todo)	17404 (todo)	7:18[a]
		4720 (seleccionado)	1:19
	12 (3 niveles)	2240 (todo)	0:43
		704 (seleccionado)	0:22
Flattening	64 (todo)	11412 (todo)	3:46
		3888 (seleccionado)	1:01
	12 (3 niveles)	1560 (todo)	0:28
		624 (seleccionado)	0:19

[a] Extraído de [PD08a]

Cuadro 3.1: Tiempos de ejecución

- Así como el método anterior se ajustaba a la generación de invariantes para elementos individuales, este produce fácilmente invariantes que se aplican a todos los elementos de la estructura original. Por ejemplo, si necesitamos probar que todos los elementos vienen de Google o MSN.

- Se sigue siendo pudiendo producir invariantes para algunos elementos de las variables unidimensionales, pero sólo aquellos escogidos por la heurística de Daikon. No hay garantía de que se generen invariantes con los elementos que nos interesen.

- Se pierde la estructura de árbol original, por lo que no se puede aplicar la información XML Schema directamente.

- Por otro lado, el número de variables se mantiene constante independientemente de los niveles o elementos anidados, lo que soluciona el problema del coste de la solución anterior. Es por eso que esta opción será la seleccionada por defecto.

Comparativa

En la tabla 3.1 se comparan los tiempos de ejecución y el número de variables obtenidos para cada solución, usando 7 casos de prueba para cada combinación.

Como se esperaba, los tiempos para la técnica de *flattening* son mejores que para *slicing*. Además podemos ver que restringir el análisis a las variables y puntos de programa relevantes reduce de forma considerable el tiempo empleado.

Las pruebas se ejecutaron en un equipo con procesador Intel Core Duo T2250 CPU, con 1GiB de memoria RAM DDR2 533MHz, y un disco duro de 80GB (5400rpm). El sistema operativo fue GNU/Linux Ubuntu 8.04.1, con el kernel 2.6.24-18-generic.

Se muestra a continuación un ejemplo de cómo varían los invariantes generados, y por tanto la información obtenida, en función de la solución empleada.

Slicing

```
(...).result[1].from[]
  one of { [Google], [MSN] }
(...).result[2].from[]
  elements one of { "Google", "MSN" }
(...).result[3].from[]
  elements one of { "Google", "MSN" }
(...).result[4].from[]
  elements == "MSN"
(...).result[5].from[]
  elements == "Google"
(...).result[6].from[]
  elements == "MSN"
```

Flattening

```
(...).result.from[]
  elements one of { "Google", "MSN" }
```

3.3.2. Implementación

Como ya se ha explicado, esta fase debe separarse en dos grandes bloques: la conversión de las trazas de ejecución al formato de entrada de Takuan -aplicando la técnica de conversión de vectores multidimensionales elegida -, y la inferencia de invariantes. Además, existen una serie de optimizaciones [Pal08].

Será un script Perl (analizadorDaikon.pl) el encargado de dirigir todo este proceso, realizando llamadas a los métodos y ficheros correspondientes. Los pasos seguidos por este script son:

1. Recibe como parámetros las trazas de ActiveBPEL, y las opciones de ejecución (como, por ejemplo, el método de aplanamiento a emplear)
2. Se ejecuta el fichero convertidorTrazas.pl, quien realizará la conversión de las trazas
   a) Este script aplica una serie de expresiones regulares para buscar los mensajes generados por las instrucciones XPath especiales que se introdujeron durante la fase de instrumentalización.
3. Se obtiene la información sobre la comparabilidad
4. Se aplica el método de aplanamiento escogido - *flattening* o *slicing* - a las **declaraciones** de los tipos.
   - Si se ha escogido *flattening*, el script aplanarDeclsIndexes.pl procesará las declaraciones.
   - Si se ha escogido *slicing*, será el script aplanarDeclsNodeSet.pl el ejecutado.
5. Se aplica el método de aplanamiento escogido a las trazas previamente convertidas

- Si se ha escogido *flattening*, se ejecutará el script aplanarDTraceIndexes.pl
- Si se ha escogido *slicing*, se ejecutará el script aplanarDTraceNodeSet.pl

6. Sobre las trazas de ActiveBPEL se ejecuta el script cobertura.pl para obtener la cobertura de ramas y de instrucciones. Esta funcionalidad forma parte de las ampliaciones realizadas sobre Takuan, por lo que se tratará con más detalle en el capítulo 8.

7. Por último, se ejecutará Daikon con toda la información preprocesada por los scripts. La salida de Daikon será la salida de todo el sistema, aunque se contempla la posibilidad de hacer cierto procesado de esa salida previamente: conversión a otro formato, inclusión de invariantes en el BPEL original para documentar, etc.

3.3.3. Resumen

Entradas Trazas de ejecución

Salidas Invariantes inferidos

Fase previa Fase de ejecución

Fase siguiente Salida suministrada al usuario

Tecnologías empleadas Daikon, Perl

3.3.4. Ejemplo

En el ejemplo 3.9 se ha recortado el identificador del punto de programa.

Listing 3.9: Ejemplo de invariantes obtenidos

```
1  ==========================================
2  marketplace...Switch_else1_sequence1:::EXIT
3  negotiationOutcome.outcome == "Deal Failed"
4  ==========================================
5  marketplace...Switch_if-condition1_sequence1:::ENTER
6  buyerInfo.offer one of { 150, 190 }
7  sellerInfo.askingPrice one of { 100, 190 }
8  ==========================================
9  marketplace....Switch_if-condition1_sequence1:::EXIT
10 buyerInfo.offer == orig(buyerInfo.offer)
11 sellerInfo.askingPrice == orig(sellerInfo.askingPrice)
12 buyerInfo.offer one of { 150, 190 }
13 negotiationOutcome.outcome == "Deal Successful"
14 sellerInfo.askingPrice one of { 100, 190 }
```

En 3.9 podemos ver tres invariantes:

1. En la salida de la secuencia principal, en la rama else del condicional, el valor de negotiationOutcome.outcome siempre será “Deal Failed” al finalizar la ejecución de la rama.

2. En la rama del if, la oferta del comprador es o 150, o 190, y el comprador siempre pide o 100, o 190
3. Al salir de la rama del if, los valores de la oferta y la demanda no han cambiado
4. Al terminar la rama del if, la respuesta (negotiationOutcome.outcome) es siempre "Deal Successful"

Aunque esto es sólo un ejemplo, conviene hacer notar que algunos de los invariantes reflejados son "falsos invariantes", producidos por un conjunto de casos de pruebas que no es lo suficientemente extenso. Esto se hace evidente con los valores de la oferta y la demanda de precios, que obviamente no están restringidos a esos dos valores, pero que son los que aparecen en los casos de prueba.

Parte II

Ampliaciones

Capítulo 4

Calendario

Gran parte del tiempo dedicado a este proyecto ha sido destinado a formación en las técnicas usadas en Takuan y en aprendizaje de las herramientas que usa internamente: BPEL, Daikon, Java, XSLT, ...

4.1. Reuniones y seminarios

Además de las reuniones con el director del proyecto, también se asistió a los seminarios que se realizaban regularmente dentro del grupo SPI&FM. Durante esas reuniones, se comentaban o exponían los avances realizados y se acordaba la dirección a seguir en adelante.

Viernes 7 de noviembre de 2008 Se acordó buscar más ejemplos de composiciones WS-BPEL para poder probar y extender Takuan

Viernes 28 de noviembre de 2008 Presentación de los ejemplos recopilados

Viernes 9 de enero de 2009 Se decidió descartar al uso de DBEStudio por problemas de estabilidad (véase apéndice C.1)

Viernes 16 de enero de 2009 Presentación de los avances realizados hasta ese momento. Se decidió que lo siguiente a abarcar sería la instrumentalización de propiedades

Viernes 23 de enero de 2009 Para esta reunión se solucionó un conflicto en la declaración de variables (6.2)

Viernes 13 de febrero de 2009 Se decidió contactar con ActiveVOS, la empresa desarrolladora de ActiveBPEL, para solicitar información sobre el formato del fichero PDD.

Martes 3 de marzo de 2009 Crear un borrador de la presentación para las IV Jornadas de Software Libre y Conocimiento Abierto de la Universidad de Cádiz (JOSLUCA4)

Martes 10 de marzo de 2009 Se ensayó la presentación para las Jornadas

Martes 31 de marzo de 2009 Se discutió sobre el ejemplo Shipping, sobre el que se debía trabajar para lograr hacerlo funcionar en Takuan

Martes 14 de abril de 2009 Takuan ya acepta propiedades y se logra hacer funcionar el ejemplo de Shipping síncrono. Se propone realizar una interfaz gráfica para NetBeans

Martes 21 de abril de 2009 Se presenta el prototipo del asistente de NetBeans y se acuerdan algunas mejoras

Martes 5 de mayo de 2009 Se acuerda añadir a Takuan la posibilidad de generar información sobre cobertura de ejecución

Martes 26 de mayo de 2009 Primera versión funcional del plugin para NetBeans

Martes 2 de junio de 2009 Se decide presentar un artículo sobre la cobertura de ramas en Takuan

Martes 9 de junio de 2009 Se decide el título y la temática de dicho artículo [Pal09b].

4.2. Diagrama de Gantt

En la figura 4.1 puede ver el diagrama de Gantt de este proyecto.

4.3. Artículos

A continuación se numeran los artículos resultantes del trabajo de este proyecto:

4.3.1. JISBD 2009

Título Takuan: generación dinámica de invariantes en composiciones de servicios web con WS-BPEL [Gar09]

Autores Antonio García Dominguez, Manuel Palomo Duarte, Inmaculada Medina Bulo y Alejandro Álvarez Ayllon

Enviado 27 de abril de 2009

Aceptado 10 de junio de 2009

4.3.2. JSWEB 2009

Título Los casos de prueba en la generación dinámica de invariantes en composiciones de servicios web con WS-BPEL [Ál09].

Autores Alejandro Álvarez Ayllón, Antonio García Domínguez, Manuel Palomo Duarte e Inmaculada Medina Bulo

Enviado 14 de junio de 2009

Aceptado 31 de julio de 2009

4.3.3. PRIS 2009

Título La cobertura de los casos de prueba en la generación dinámica de invariantes en composiciones WS-BPEL [Pal09b].

Autores Manuel Palomo Duarte, Alejandro Álvarez Ayllón, Antonio García Domínguez e Inmaculada Medina Bulo

Enviado 26 de junio de 2009

Aceptado 21 de julio de 2009

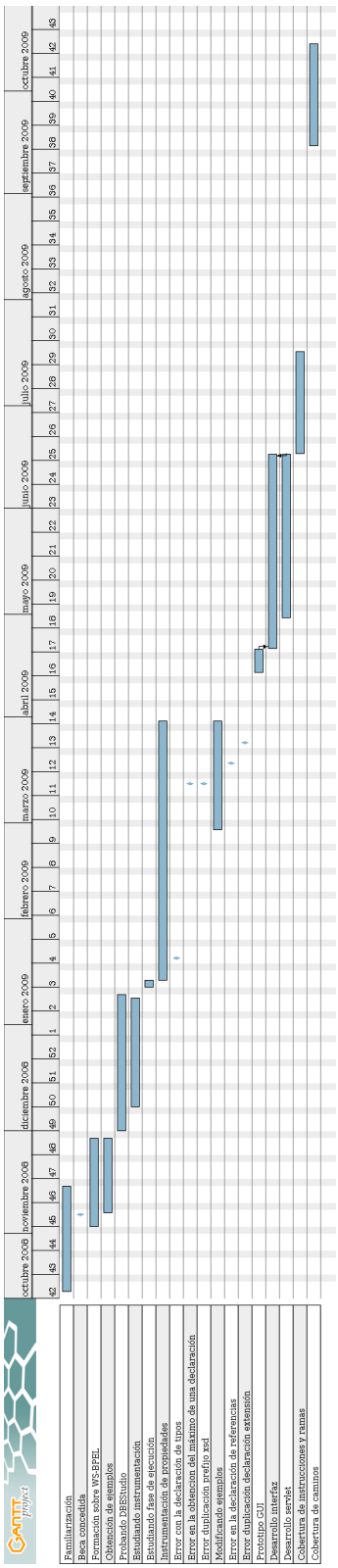

Figura 4.1: Diagrama de Gantt

Capítulo 5

Obtención de nuevos ejemplos

Una de las limitaciones de trabajar sobre un lenguaje tan reciente como WS-BPEL es que apenas existen ejemplos reales, o al menos realistas, sobre los que trabajar. La mayoría de los autores de tesis y artículos se limitan a trabajar sobre ejemplos clásicos como el del Préstamo bancario incluido en el estándar [Com07].

Por tanto, la primera actividad necesaria para poder empezar a trabajar sobre ampliaciones de Takuan era encontrar nuevos ejemplos y comprobar si funcionaban con la versión de disponible en ese momento.

A continuación se enumeran los ejemplos que se encontraron, su origen, y qué reformas exigieron - en algunos casos - para poder funcionar en Takuan. Algunos de ellos están disponibles en un repositorio público[1].

5.1. Shipping

El ejemplo Shipping procede de la documentación del estándar OASIS [OAS07]. Como se ve en 5.1, es relativamente simple. Básicamente implementa un servicio de envío de paquetes. A pesar de su simplicidad, sacó a relucir dos posibles puntos débiles de Takuan:

Asíncronía Este proceso es asíncrono, y Takuan no había sido probado con procesos de este tipo. Finalmente, resultó funcionar sin necesidad de realizar modificaciones, aunque hubo que averiguar como configurar BPELUnit para que funcionara. Básicamente fue necesario emplear bloques `sendReceiveAsynchronous` en lugar del habitual `sendReceive`.

Propiedades En WS-BPEL las variables pueden tener asignadas propiedades, que no son sino "alias" de elementos de la variable, y Takuan no era capaz de trabajar con ellas. Cómo se añadió esta funcionalidad puede consultarse en el capítulo 7. Más información sobre las propiedades, en el apéndice B sobre WS-BPEL.

Además de la versión asíncrona, se creó una versión modificada síncrona.

[1] `http://neptuno.uca.es/redmine/projects/show/wsbpel-comp-repo`

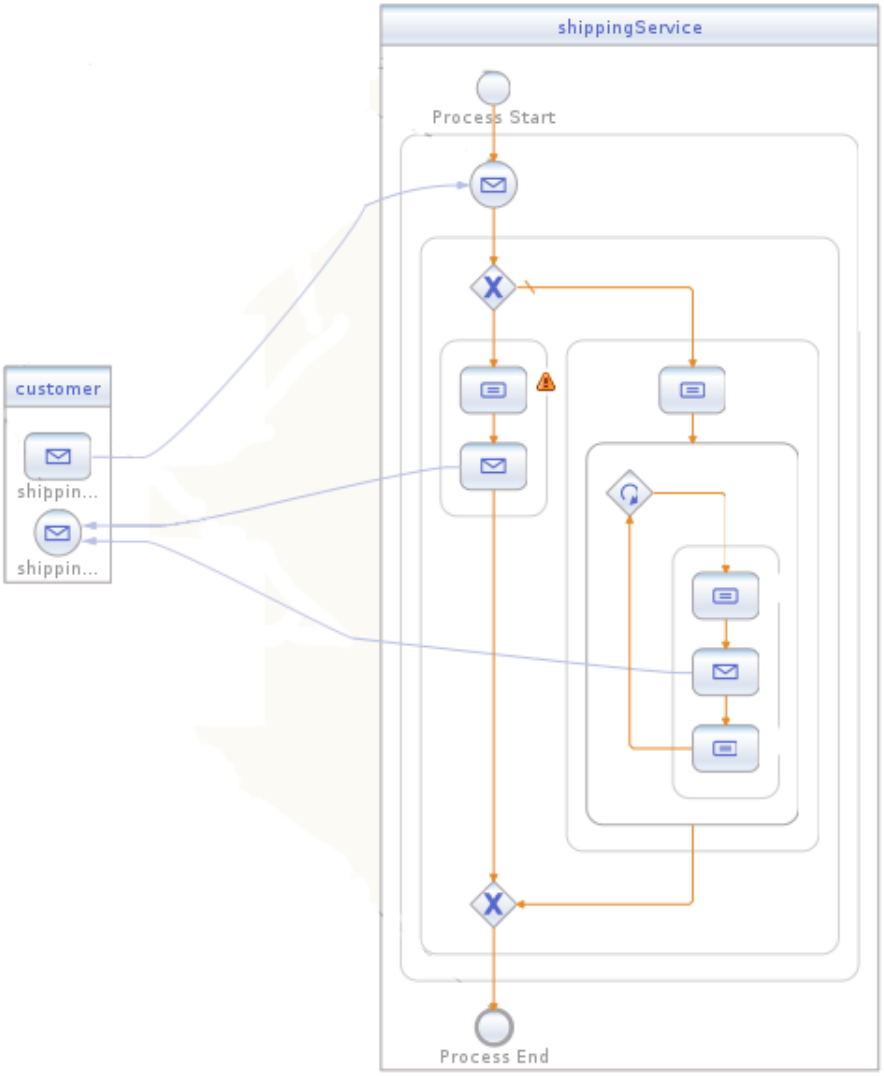

Figura 5.1: Ejemplo Shipping

5.2. Auction

Al igual que Shipping, procede del estándar WS-BPEL 2.0. También es asíncrono, y tampoco funcionaba en Takuan. En este caso, el problema se debe a la inclusión de operaciones que permiten asignar directamente los valores de un socio (*partnerLink*) a una variable, y viceversa.

Actualmente esta carencia no está todavía solucionada.

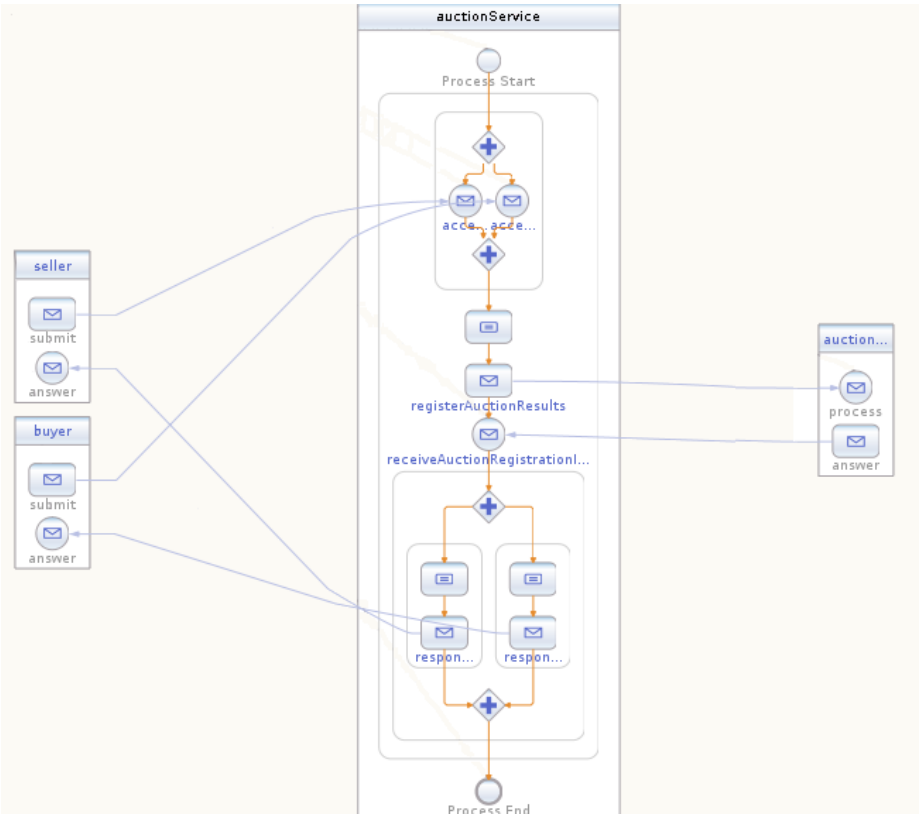

Figura 5.2: Ejemplo Auction

5.3. Ordering

Procede también del estándar OASIS. Hace un uso intensivo de elementos opacos: elementos que no especifican las operaciones concretas que se realizan en el servicio real. Es decir, es un modelo abstracto y, por tanto, inútil para nuestros fines.

5.4. Complex Data Exchange

Procede de la web de ActiveVOS [Act08]. Es un ejemplo incompleto y escaso, que no resultó de utilidad.

5.5. Market Place

Procedente también de la web de ActiveVOS. Es un proceso síncrono que trabaja con dos peticiones: la del comprador y la del vendedor.

Básicamente su funcionamiento es enviar un mensaje de éxito si el precio pedido por el vendedor es menor o igual a la cantidad ofrecida por el comprador.

5.6. Travel Reservation Service

Extraído de la web de NetBeans [Mica]. Es un ejemplo completamente definido con enlaces tanto síncronos como asíncronos. Funcionaba en Takuan desde un principio.

No se incluye una imagen de este proceso por su tamaño.

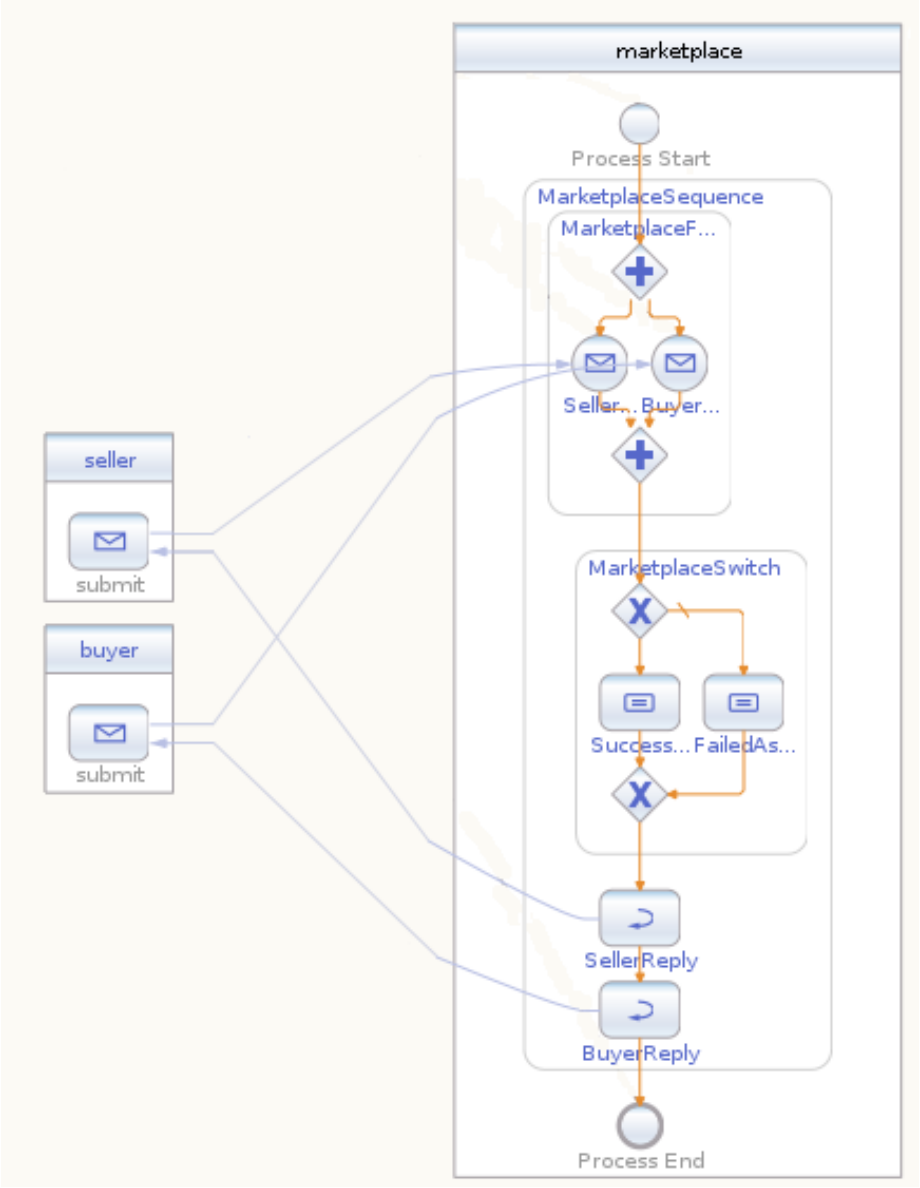

Figura 5.3: Ejemplo MarketPlace

Capítulo 6

Errores corregidos

Durante el análisis de Takuan se descubrieron varios errores que fueron subsanados.

6.1. Error en el procesado de referencias en XML-Schema

Un error en la implementación del instrumentador provocaba errores cuando un fichero XML Schema, al declarar un elemento, hacía referencia a otro:

```
1 <xsd:element ref="tns:shipComplete"/>
```

Se corrigió para que al generar el fichero con todos los tipos reunidos, el instrumentador fuera capaz de seguir estas referencias.

6.2. Conflicto en la declaración de variables

El ejemplo Shipping declaraba algunas variables como pertenecientes a un tipo XML Schema complejo, empleando para ello el atributo "type". En la versión inicial esto provocaba un conflicto interno en hojaDecls.xslt, fallando el proceso de instrumentación.

Veamos un ejemplo de cómo se declara un nuevo tipo de datos dentro de una hoja WSDL:

```
1 <wsdl:definitions targetNamespace="http://example.com/
      shipping/interfaces/">
2 <wsdl:types>
3   <xsd:schema
4     targetNamespace="http://example.com/shipping/ship.xsd
        ">
5     <xsd:simpleType name="itemCountType">
6       <xsd:restriction base="xsd:int">
7           <xsd:minInclusive value="1" />
8           <xsd:maxInclusive value="50" />
9       </xsd:restriction>
```

```
10      </xsd:simpleType>
11    </xsd:schema>
12  </wsdl:types>
13  </wsdl:definitions>
```

Por un error en hojaTipos.xslt, se tomaba el espacio de nombres objetivo del documento WSDL, no de la declaración XML Schema, al buscar los tipos asociados a través del atributo "type".

Se parcheó el error de forma que ahora acepta variables complejas declaradas a través del atributo "type".

6.3. Asignaciones no soportadas

Al realizar pruebas con el ejemplo Shipping se detectó que Takuan no era capaz de instrumentar correctamente el uso de propiedades, generando código sintácticamente correcto, pero erróneo.

Cómo se solucionó este error se detalla en el capítulo 7.

6.4. Error al obtener el máximo de un entero

Por un error en el código del instrumentalizador, no se obtenía correctamente el valor máximo posible para una variable entera. El error era muy simple, simplemente se cambió en el fichero hojaDecls.xslt

```
1  <maxvalue>
2    <xsl:value-of select="max(($restriccionesPropias/
         maxvalue,
3  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣$restriccionesBase/maxvalue)
       )"/>
4  </maxvalue>
```

Por

```
1  <maxvalue>
2    <xsl:value-of select="min(($restriccionesPropias/
         maxvalue,
3  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣$restriccionesBase/maxvalue)
       )"/>
4  </maxvalue>
```

6.5. Duplicación del prefijo xsd

En este caso al generar la información sobre los tipos, se duplicaba el prefijo "xsd" de XML Schema cuando una variable era declarada de un tipo base. Se modificó el fichero hojaDecls.xslt, cambiando

```
1  <xsl:otherwise>
2    <xsl:value-of select="concat('xsd:',␣@base)"/>
3  </xsl:otherwise>
```

Por

```
<xsl:otherwise>
  <xsl:choose>
    <xsl:when test="not(contains(@base, ': '))">
      <xsl:value-of select="concat('xsd:', @base)" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="@base" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:otherwise>
```

Es decir, se asegura de que el tipo base no venga ya con prefijo.

6.6. Duplicación de la declaración del espacio de nombres

El error en este caso se encontraba en el fichero hojaInspeccionar.xslt, donde se añadía el bloque de información sobre el espacio de nombres de la Universidad de Cádiz al proceso inspeccionado sin cuidar de si estaba o no ya incluido. Un simple cambio de:

```
<bpel:extension mustUnderstand="no" namespace="{$uca}" />
```

Por:

```
<xsl:if test="not(bpel:extension[@namespace = $uca])">
  <bpel:extension mustUnderstand="no" namespace="{$uca}"
      />
</xsl:if>
```

Capítulo 7

Instrumentación de propiedades

Como ya se ha comentado con anterioridad, dentro del ejemplo de Shipping se realizan asignaciones como la siguiente:

```
1 <copy>
2   <from variable="shipRequest"
3         property="props:shipOrderID" />
4   <to variable="shipNotice"
5         property="props:shipOrderID" />
6 </copy>
```

Lo que provoca que Takuan falle al realizar la instrumentalización con el siguiente mensaje de error:

```
An empty sequence is not allowed as the first argument
```

Ese error se debe a que el instrumentador espera que el elemento from de la asignación contenga algo en su interior, ya sea un literal o el nombre de una variable.

La solución que se pensó inicialmente fue:

- Generar variables "virtuales", nombrándolas de la forma _variable_propiedad
- Generar sus equivalentes dummies para realizar la instrumentación de la forma habitual
- En cada asignación desde propiedades, realizar primero una copia del valor de la propiedad a la variable "virtual"
- Realizar la instrumentalización de la variable virtual
- Realizar la asignación original

Sin embargo, esta medida genera mucho código, dificultando la legibilidad del proceso resultante, y haciéndolo más pesado.

Así que tras discutirlo, se acabó por optar por otra solución que pasa por añadir nuevas extensiones XPath:

- Generar las variables dummies siguiendo el esquema dummy_variable_propiedad
- En cada asignación desde propiedades, instrumentarla de la forma habitual, pero empleando la nueva extensión uca:inspeccionarPropiedad, en lugar del original uca:inspeccionar.
- En cada comparación, condición, ..., sustituir las llamadas a bpel:getVariableProperty por uca:ambitoPropiedad, que actúa como uca:ambito. Es decir, genera información en el log de ejecución para poder trazar los ámbitos.

Pese a no declararse explícitamente, en la salida de la traza las propiedades siguen siendo referidas por variables "virtuales" que siguen el esquema _variable_propiedad.

Así, la transformación realizada sobre la asignación que hemos visto antes generaría para obtener el ámbito:

```
1 <copy>
2   <bpel:from>
3     uca:ambitoPropiedad('shipRequest','props:shipOrderID'
          )
4   </bpel:from>
5   <to variable="shipNotice" property="props:shipOrderID"/
      >
6 </copy>
```

Y para inspeccionar el valor, tanto al comienzo como al final de un bloque:

```
1 <bpel:copy>
2   <bpel:from>
3     uca:inspeccionarPropiedad('shipRequest', '
        props:shipOrderID')
4   </bpel:from>
5   <bpel:to>
6       $dummy_shipRequest_shipOrderID
7   </bpel:to>
8 </bpel:copy>
```

Tras implementar estas dos extensiones en Java, la salida resultante en la traza es:

```
[1][2009-10-21 16:31:54.416] :
  INSPECTION($shipRequest~props:itemsCount) = 1 []
```

Capítulo 8

Cobertura

8.1. Motivación

Como ya se ha mencionado con anterioridad, a la hora de realizar un pruebas sobre un sistema - ya sea para obtener sus invariantes o simplemente para comprobar su corrección - se busca que el conjunto de casos de prueba sea lo más amplio y representativo posible.

Los datos sobre la cobertura de un conjunto de casos de prueba, es decir, sobre las instrucciones, ramas y caminos cubiertos, nos ofrece un parámetro de la calidad de dicho conjunto. Cuanto mayor sea la cobertura, mayor la representatividad, y mayor la calidad de las pruebas.

Se puede definir el criterio de cobertura como [Gas05, Ram07]:

- Como un criterio de adecuación, que establece una serie de reglas usadas para determinar si el conjunto de casos de prueba es completo o no
- Como un criterio de selección, para seleccionar un conjunto de casos de pruebas para un programa y especificación dado

En definitiva, el objetivo de ser capaces de obtener información de cobertura es permitir al usuario de Takuan el mejorar su conjunto de casos de prueba, de forma que los invariantes obtenidos sean más representativos.

8.2. Tipos de cobertura

8.2.1. Cobertura de instrucciones

La cobertura de sentencias (también conocida como cobertura de instrucciones) es la más simple: nos asegura que un conjunto de casos de prueba ejecuta todas las instrucciones del programa al menos una vez [Pal09b].

8.2.2. Cobertura de ramas

Para cumplir con la cobertura de ramas, un conjunto de casos de prueba tiene que tomar al menos una vez todas las ramas de cada instrucción condicional.

Tener una cobertura de ramas completa implica, necesariamente, ejecutar todas las instrucciones del programa, por lo que aquel conjunto que cumpla la cobertura de ramas cumple igualmente la de instrucciones.

Sin embargo, tener cobertura de instrucciones no implica tener cobertura de ramas:

```
hacer_algo
if x == 5 then
  cuerpo_condicion
hacer_algo
```

En el ejemplo anterior, si tuviéramos un único caso de prueba donde x es igual a 5, tendríamos cobertura de instrucciones - todas las sentencias del código se han ejecutado al menos una vez - sin embargo, no se ha cubierto la rama *else* (implícita) del condicional.

8.2.3. Cobertura de caminos

Por último la ejecución de caminos es la más completa, pues obliga a ejecutar toda combinación de ramas que se puedan dar en el programa, lo que implica ejecutar todas las ramas condicionales y, por tanto, todas las instrucciones. Esta cobertura es de las más completas que se definen, pero sólo puede conseguirse en programas sin bucles [Pal09b].

```
si x > 1
  x=8
si y > 1
  y=0
```

Para ramas basta con x=2,y=2 e x=1,y=1 Y para caminos hacen falta también x=1,y=2 e x=2,y=1

8.3. Implementación en Takuan

Entre la ejecución del conjunto de casos de prueba y el procesado de Daikon, Takuan primero ejecuta una serie de scripts Perl que realizan un preprocesado de las trazas, adaptándolas a Daikon, y extrayendo información sobre la comparabilidad.

Para implementar la obtención de información sobre la cobertura se ha añadido un script a esta fase llamado `cobertura.pl`. Este script, llamado desde el "director" `analizadorDaikon.pl`, recibe como parámetros todos los ficheros log generados por la ejecución del proceso de negocio e imprime por la salida estándar - redirigida por el script director - información sobre:

Instrucciones ejecutadas alguna vez Junto a una fracción del tipo <número de veces ejecutada>/<número de casos de prueba>

Instrucciones no ejecutadas nunca

Ramas ejecutadas alguna vez

Ramas nunca ejecutadas

8.3.1. Cobertura de instrucciones

Las trazas de ejecución tienen el siguiente formato:

```
1  [1][2009-06-18 11:59:40.635] : Executing [/process]
2  [1][2009-06-18 11:59:40.636] : Executing
3    [/process/sequence[@name='MarketplaceSequence']]
4  [1][2009-06-18 11:59:40.636] : Executing
5    [/process/sequence[@name='MarketplaceSequence']/flow[
         @name='MarketplaceFlow']]
6  [1][2009-06-18 11:59:40.638] :
7    Executing [/process/.../receive[@name='SellerReceive']]
8  [1][2009-06-18 11:59:40.656] :
9    Completed normally [/process/.../receive[@name='
         SellerReceive']]
10 [1][2009-06-18 11:59:40.656] :
11   Executing [/process/.../receive[@name='BuyerReceive']]
12 [1][2009-06-18 11:59:42.394] : Completed normally
13 [1][2009-06-18 11:59:42.454] :
14   INSPECTION($sellerInfo.inventoryItem) = Takuan []
```

Y tenemos una para cada caso de prueba ejecutado. Como podemos ver, extraer la información sobre las instrucciones ejecutadas alguna vez es relativamente sencillo:

1. Para cada traza pasada como argumento
   a) Se mete en un array las instrucciones precedidas por `Executing`, es decir, las ejecutadas.
   b) Se mete en otro array las instrucciones precedidas por `Will not execute`
2. A partir del array de instrucciones ejecutadas de cada caso de prueba, generamos la lista global de instrucciones ejecutadas alguna vez. Se incrementa el contador de número de veces ejecutadas cada vez que aparece.
3. Para cada instrucción contenida en el array de instrucciones no ejecutadas de cada caso de prueba, si no se encuentra en el array general de ejecutados, nunca se ha ejecutado.
4. Imprimir los resultados

La salida para esta parte sigue el formato siguiente:

```
1  Instruction coverage
2  ====================
3
4  Executed 35/35 (100%)
5  ---------------------
6  (2/2) /process/.../if-condition/sequence/assign[3]/copy
       [2]
7  (2/2) /process/sequence[@name='MarketplaceSequence']/flow
       [@name='MarketplaceFlow']
```

```
8  (2/2) /process/.../if-condition/sequence/assign/copy[2]
9  (2/2) /process/.../if-condition/sequence/assign/copy[0]
10 (2/2) /process/sequence[@name='MarketplaceSequence']/
       reply[@name='BuyerReply']
11 (2/2) /process/sequence[@name='MarketplaceSequence']/
       assign/copy[4]
12 [...]
13 (2/2) /process/.../if-condition/sequence/assign/copy[3]
14 (2/2) /process/.../if-condition/sequence/assign[3]/copy
       [4]
15 (2/2) /process/sequence[@name='MarketplaceSequence']/
       assign[2]/copy[3]
16 (2/2) /process/sequence[@name='MarketplaceSequence']/if[
       @name='MarketplaceSwitch']
17 (2/2) /process/sequence[@name='MarketplaceSequence']
18 (2/2) /process/sequence[@name='MarketplaceSequence']/
       assign/copy[0]
19
20 Never executed 0/35 (0%)
21 ----------------------------
```

8.3.2. Cobertura de ramas

La extracción de cobertura de ramas es ligeramente más compleja, ya que hay que encontrar las ramas a partir de las instrucciones, y "generar" los elses implícitos.

Por desgracia, la versión actual sólo funciona con condicionales simples. Para el futuro, se quiere añadir soporte para bucles.

Los pasos son los siguientes:

1. A partir de las instrucciones ejecutadas alguna vez, y aplicando expresiones regulares, extraer las ramas ejecutadas alguna vez. Tanto if como elses.

2. A partir de las instrucciones nunca ejecutadas, extraer las ramas nunca ejecutadas. Tanto if, como elses.

3. En este momento, tenemos información de cobertura para las ramas explícitas en el código, pero es necesario buscar los elses implícitos para obtener una información más completa.

4. Por cada caso de prueba

   a) Para cada if no ejecutado en este caso de prueba

      1) Generamos la expresión XPath del else correspondiente
      2) Comprobamos que el elemento padre se ha ejecutado - necesario si se quiere funcionar con condicionales anidados.
      3) Si el padre se ejecutó, añadimos el else a la lista de ejecutados si no estaba ya [1]

[1] Si el elemento padre se ha ejecutado, y no se entró por el if, entonces se entró por el else

4) Si el padre no se ejecutó, se añade a la lista de nunca ejecutadas si no está en la de ejecutadas. Esto es necesario porque puede ser un else implícito y no estar dentro de no ejecutados.

5. Ahora tenemos los if ejecutados, los nunca ejecutados, los elses ejecutados alguna vez y los no ejecutados nunca **porque el padre no se ha ejecutado**. Necesitamos encontrar los elses que nunca se han ejecutado porque siempre se ha entrado por la rama verdadera del condicional.

6. Para cada if ejecutado alguna vez

7. Comprobamos cuántas veces se ha ejecutado el nodo padre, de haberlo

8. Si el número de veces que se ha ejecutado el if es igual al número de veces que se ha ejecutado el padre, entonces nunca se ha entrado por la rama else - esté explícitamente definida o no.

La salida para la cobertura de ramas se imprime a continuación de la de instrucciones. A continuación se muestra la información de cobertura de ramas para el mismo ejemplo que el anterior:

```
1  Branches coverage
2  ================
3
4  Branches executed 1 / 2 (50%)
5  ------------------------------
6  /process/sequence[@name='MarketplaceSequence']/if[@name='
       MarketplaceSwitch']/
7    if-condition
8
9  Branches not executed 1 / 2 (50%)
10 ----------------------------------
11 /process/sequence[@name='MarketplaceSequence']/if[@name='
       MarketplaceSwitch']/
12   else
```

En este ejemplo además podemos ver cómo hay una cobertura del 100 % de instrucciones, pero sólo del 50 % de ramas. Esto se debe a que la rama else existe implícitamente, aunque no aparezca en el código, y nunca se entra por ella.

Conviene hacer notar que a día de hoy la cobertura de ramas no funciona para bucles ni para actividades específicas de WS-BPEL como `pick`.

8.3.3. Cobertura de caminos

Este es el método más general y completo de todos - una cobertura de caminos del 100 % implica una cobertura del 100 % de ramas y, por lo tanto, de instrucciones - pero también el más complejo.

A diferencia de los dos métodos anteriores no es suficiente con tener las trazas de ejecución para poder obtener la información de cobertura de caminos. También es necesario disponer del fichero BPEL instrumentalizado para generar todos los caminos posibles, y compararlos con los realmente ejecutados.

Obtener los caminos ejecutados se realiza de la misma forma que en los casos anteriores: procesando las trazas buscando las líneas que comiencen con "Executing". Sin embargo, obtener todos los posibles caminos requiere un algoritmo más complejo.

Obtención de todos los caminos posibles

La obtención de todos los caminos posibles se obtiene siguiendo el siguiente algoritmo:

1. Se parsea el documento XML obteniendo una representación en forma de árbol
2. Se mete en la pila el nodo raíz del código (`/process/sequence`)
3. Se extrae el nodo del tope de la pila y se mete el siguiente nodo (*sibling*)
4. Si el nodo es un nodo normal, se añade a la lista de sentencias
5. Si el nodo es un condicional
   a) Se duplica la pila y la lista de sentencias
   b) Se añade a la pila original el nodo correspondiente a la rama del condicional
   c) Se añade a la pila copiada el nodo correspondiente a la rama del else
   d) Se crea un nuevo hilo con la pila copiada, comenzando por el paso 3 de nuevo
6. Si quedan nodos en la pila, volvemos al paso tres
7. Por último, cuando la pila se vacía, añadimos a la lista de posibles camino la lista de secuencias "ejecutadas"

Este proceso puede verse gráficamente en la figura 8.1. Se utiliza un esquema de colores para distinguir entre el hilo original, y el hilo clonado.

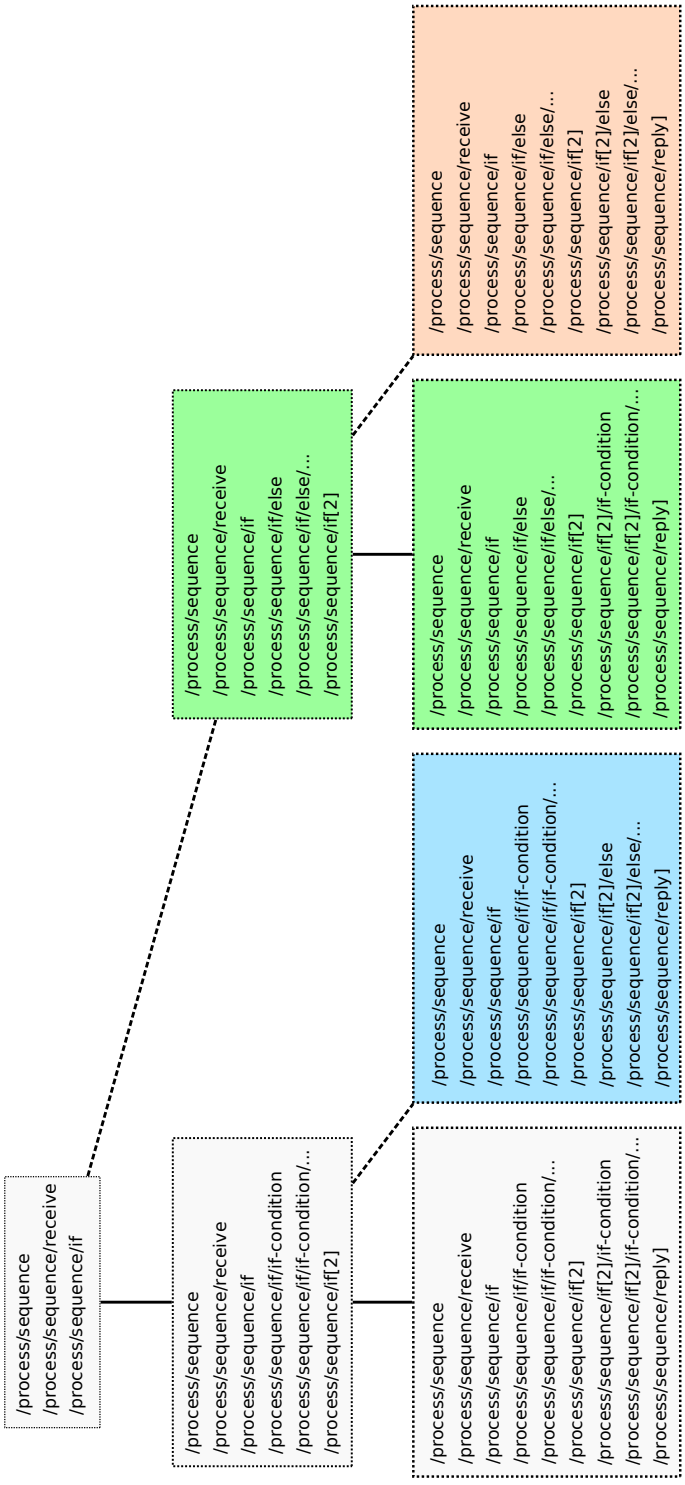

Figura 8.1: Obtención de los posibles caminos

Comparación

Una vez disponemos de todos los caminos posibles, y la lista de los caminos realmente ejecutados, sólo resta hacer una comparación, buscando cada camino posible en la lista de ejecutados, de forma que cuando uno no se encuentra, es notificado.

Conviene hacer notar que un camino no tomado no implica, necesariamente, una carencia en los casos de prueba, ya que el resultado de un condicional puede influir en otro. Por ejemplo:

```
if x > 0 then
  y = 1
else
  y = -x;
fi

if y == -2 then
  ...
else
  ...
fi
```

Nunca se entrará por la rama if del primer condicional y por la del segundo en una misma ejecución, por lo que ese camino nunca se ejecutará.

Ejecución

Este subprograma, además de incluirse en Takuan, se distribuye como un fichero JAR, por lo que se ejecuta de la forma

```
$ java -jar CoberturaCaminos.jar [opciones] <BPEL> <Lista de trazas>
```

Las opciones disponibles son

-h Muestra la ayuda

-d Imprime información de depuración

-o <**fichero**> Imprime la información sobre los caminos no ejecutados en el fichero indicado. Si no se especifica, se imprimirá en la salida estándar.

-p <**fichero**> Imprime la información sobre todos los caminos posibles en el fichero indicado.

-e <**fichero**> Imprime la información sobre los caminos ejecutados en el fichero indicado.

–**plain** Por defecto, el programa genera la información en formato XML. Si se especifica esta opción, la salida será texto plano.

Ejemplo

A continuación se muestra un ejemplo de todos los caminos posibles para una composición BPEL. La salida ha sido generada en texto plano, aunque el programa genera por defecto un documento XML.

El formato es similar para la salida de los caminos ejecutados, y los no ejecutados:

```
1  Maximum length of an execution path: : 46
2  Possible execution paths: 2
3  Minimum length of an execution path: : 36
4
5  ============================
6
7  Execution path no.1
8  36 sentences
9
10 /process
11 /process/sequence
12 /process/sequence/receive
13 /process/sequence/assign
14 /process/sequence/assign/copy[0]
15 /process/sequence/assign/copy[1]
16 /process/sequence/assign/copy[2]
17 /process/sequence/assign/copy[3]
18 /process/sequence/assign/copy[4]
19 /process/sequence/assign[2]
20 /process/sequence/assign[2]/copy[0]
21 /process/sequence/if
22 /process/sequence/if/if-condition
23 /process/sequence/if/if-condition/sequence
24 /process/sequence/if/if-condition/sequence/assign
25 /process/sequence/if/if-condition/sequence/assign/copy[0]
26 /process/sequence/if/if-condition/sequence/assign/copy[1]
27 /process/sequence/if/if-condition/sequence/assign/copy[2]
28 /process/sequence/if/if-condition/sequence/assign/copy[3]
29 /process/sequence/if/if-condition/sequence/assign/copy[4]
30 /process/sequence/if/if-condition/sequence/assign[2]
31 /process/sequence/if/if-condition/sequence/assign[2]/copy
       [0]
32 /process/sequence/if/if-condition/sequence/assign[2]/copy
       [1]
33 /process/sequence/if/if-condition/sequence/assign[3]
34 /process/sequence/if/if-condition/sequence/assign[3]/copy
       [0]
35 /process/sequence/if/if-condition/sequence/assign[3]/copy
       [1]
36 /process/sequence/if/if-condition/sequence/assign[3]/copy
       [2]
37 /process/sequence/if/if-condition/sequence/assign[3]/copy
       [3]
```

```
38 /process/sequence/if/if-condition/sequence/assign[3]/copy
       [4]
39 /process/sequence/assign[3]
40 /process/sequence/assign[3]/copy[0]
41 /process/sequence/assign[3]/copy[1]
42 /process/sequence/assign[3]/copy[2]
43 /process/sequence/assign[3]/copy[3]
44 /process/sequence/assign[3]/copy[4]
45 /process/sequence/reply[@name='Respuesta']
46 ================================
47
48 Execution path no.2
49 46 sentences
50
51 /process
52 /process/sequence
53 /process/sequence/receive
54 /process/sequence/assign
55 /process/sequence/assign/copy[0]
56 /process/sequence/assign/copy[1]
57 /process/sequence/assign/copy[2]
58 /process/sequence/assign/copy[3]
59 /process/sequence/assign/copy[4]
60 /process/sequence/assign[2]
61 /process/sequence/assign[2]/copy[0]
62 /process/sequence/if
63 /process/sequence/if/else
64 /process/sequence/if/else/sequence
65 /process/sequence/if/else/sequence/assign
66 /process/sequence/if/else/sequence/assign/copy[0]
67 /process/sequence/if/else/sequence/assign/copy[1]
68 /process/sequence/if/else/sequence/assign/copy[2]
69 /process/sequence/if/else/sequence/assign[2]
70 /process/sequence/if/else/sequence/assign[2]/copy[0]
71 /process/sequence/if/else/sequence/while
72 /process/sequence/if/else/sequence/while/sequence
73 /process/sequence/if/else/sequence/while/sequence/assign
74 /process/sequence/if/else/sequence/while/sequence/assign/
       copy[0]
75 /process/sequence/if/else/sequence/while/sequence/assign/
       copy[1]
76 /process/sequence/if/else/sequence/while/sequence/assign/
       copy[2]
77 /process/sequence/if/else/sequence/while/sequence/assign
       [2]
78 /process/sequence/if/else/sequence/while/sequence/assign
       [2]/copy[0]
79 /process/sequence/if/else/sequence/while/sequence/assign
       [2]/copy[1]
80 /process/sequence/if/else/sequence/while/sequence/assign
```

```
    [3]
81 /process/sequence/if/else/sequence/while/sequence/assign
    [3]/copy[0]
82 /process/sequence/if/else/sequence/while/sequence/assign
    [4]
83 /process/sequence/if/else/sequence/while/sequence/assign
    [4]/copy[0]
84 /process/sequence/if/else/sequence/while/sequence/assign
    [4]/copy[1]
85 /process/sequence/if/else/sequence/while/sequence/assign
    [4]/copy[2]
86 /process/sequence/if/else/sequence/assign[3]
87 /process/sequence/if/else/sequence/assign[3]/copy[0]
88 /process/sequence/if/else/sequence/assign[3]/copy[1]
89 /process/sequence/if/else/sequence/assign[3]/copy[2]
90 /process/sequence/assign[3]
91 /process/sequence/assign[3]/copy[0]
92 /process/sequence/assign[3]/copy[1]
93 /process/sequence/assign[3]/copy[2]
94 /process/sequence/assign[3]/copy[3]
95 /process/sequence/assign[3]/copy[4]
96 /process/sequence/reply[@name='Respuesta']
```

Capítulo 9

Interfaz gráfica para Takuan

9.1. Introducción

En el artículo que se presentó en las *Jornadas de Ingeniería del Software y Bases de Datos* dentro de apartado "Demos", se mostró la interfaz gráfica que permite usar Takuan de forma más sencilla e intuitiva. Hasta ese momento sólo se podía ejecutar Takuan o manualmente por consola, o a través de un script Ant; además de tener que escribir manualmente los atributos necesarios para la instrumentalización de un proceso (variables a inspeccionar y puntos de programa a instrumentalizar).

9.1.1. NetBeans vs Eclipse

El prototipo tenía que estar hecho en una semana, así que se estimó necesario trabajar sobre algo existente que facilitara y agilizara el diseño de una aplicación de estas características. Como la integración con un editor visual de WS-BPEL también era un valor añadido, se decidió desarrollar esta interfaz sobre Eclipse [Ecl] o NetBeans [Mica], dos IDEs con editor BPEL maduro, y con arquitectura de plugins.

Aunque en principio se quería extender el editor elegido para añadirle las extensiones de Takuan, rápidamente se descartó la idea por su complejidad, así que se optó por una interfaz tipo asistente, que guíe al usuario paso a paso.

Aunque la flexibilidad y potencia del sistema de plugins de Eclipse es mayor, NetBeans permite diseñar los nuevos plugins a través de asistentes y editores visuales de los formularios. Como el tiempo era crucial, se optó por NetBeans.

9.2. Arquitectura

A continuación se describen las partes que componen el asistente de Takuan para NetBeans.

9.2.1. Interfaz

Fue lo primero en diseñarse y programarse y, como ya se ha mencionado, se optó por una interfaz de tipo asistente, con nueve pasos:

1. Bienvenida, descripción del asistente y copyright (Figura 18.4)
2. Valores por defecto de la instrumentalización: profundidad máxima e inspección por defecto de las variables.
3. Puntos de programa a inspeccionar
4. Variables a instrumentalizar. Para cada una, puede escogerse entre Sí, No, o la configuración por defecto (figura 18.7)
5. Ficheros de los que depende el proceso, y conjunto de casos de pruebas
6. URI del servlet de Takuan. Será a este servlet al que la interfaz se conectará y enviará el proceso.
7. Diálogo de progreso
8. Recuperación de la información de invariantes y coberturas

Todos estos componentes se encuentran dentro de tres módulos Java:

es.uca.takuan.netbeans Ficheros de configuración que permiten a NetBeans cargar el plugin

es.uca.takuan.netbeans.wizard Elementos del asistente

es.uca.takuan.netbeans.components Elementos adicionales de la interfaz: botón de inicio y consola de depuración

Ninguno de estos elementos realiza ningún procesado más allá de presentar la información al usuario. El parseado y modificación del fichero BPEL, más la interacción con el servidor, se llevan a cabo por el módulo Cliente, que es completamente independiente de NetBeans.

La interfaz gráfica implementa la interfaz Java `EventListener` definida en el módulo cliente. De esta forma el Cliente puede enviar notificaciones a la interfaz.

9.2.2. Cliente

Este módulo implementa la lógica para interactuar con el servidor, y para cargar y modificar el fichero WS-BPEL. Es completamente independiente del IDE escogido, por lo que es fácilmente reutilizable por otras interfaces en el caso de que se quieran desarrollar. El módulo es `es.uca.takuan.client`, y contiene tres clases y una interfaz:

Clase Bpel Esta clase se encarga de cargar y parsear un fichero WS-BPEL. También proporciona métodos para navegar por él y modificarlo. Por último, permite volver a serializar el fichero WS-BPEL.

Clase NamespaceContext Clase privada de apoyo para BPEL. No tiene mayor interés.

Clase Remote Implementa toda la lógica de la interacción con el servidor.

Interfaz EventListener Interfaz que debe implementar la clase que vaya a interactuar con los eventos generados por la clase Remote.

En el CD adjunto se encuentra la documentación generada con JavaDoc para todas estas clases.

Diagrama de estados

En la figura 9.1 se muestra el diagrama de estados del cliente - Clase Remote. Se omiten los estados `CANCELLED` y `ERROR`.

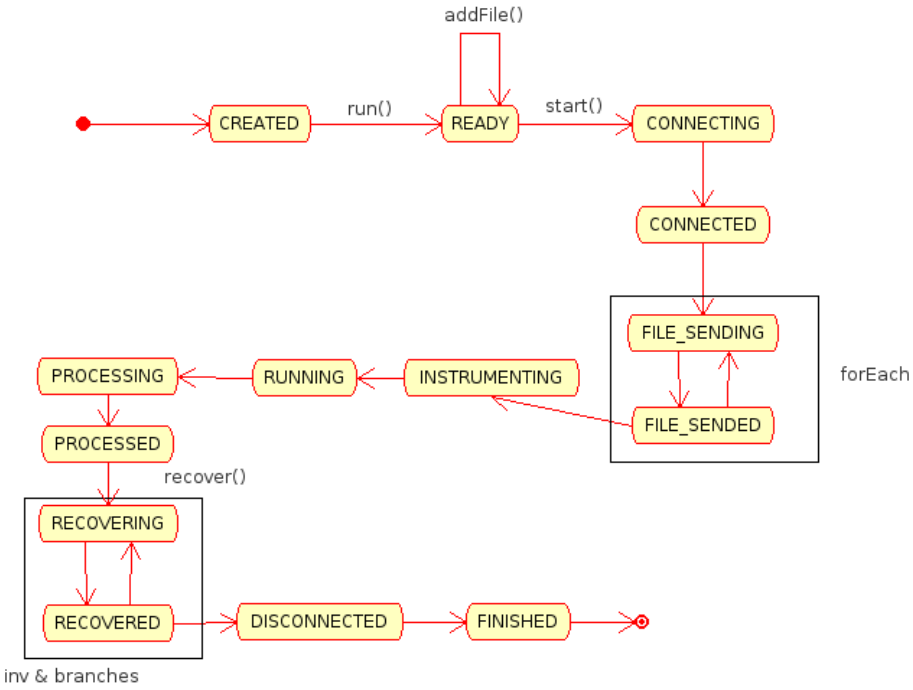

Figura 9.1: Diagrama de estados del cliente

READY La clase está creada y lista para conectarse

CONNECTING Se está realizando la conexión al servlet

CONNECTED Se ha realizado la conexión con éxito

FILE_SENDING Se está enviando un fichero

FILE_SENDED Se ha terminado de enviar un fichero

INSTRUMENTING El servidor está instrumentando el fichero

RUNNING El servidor está ejecutando el proceso inspeccionado

PROCESSING El servidor está procesando las trazas

PROCESSED El servidor ha finalizado de procesar las trazas

RECOVERING Se está recuperando un fichero desde el servidor

RECOVERED Se ha terminado de recuperar un fichero desde el servidor

DISCONNECTED Se ha finalizado la conexión

FINISHED Se ha finalizado el proceso

CANCELLED El usuario ha cancelado el proceso

ERROR Ha ocurrido un error (puede recuperarse mediante el método getErrorMessage)

9.2.3. Servidor

La interfaz se limita a facilitar el uso de Takuan, permitiendo marcar las variables y puntos del programa a instrumentar. Sin embargo, todo el trabajo de instrumentalización, ejecución y procesado de las trazas sigue siendo realizado por Takuan, aunque en este caso es un Servlet Java el que actúa de interfaz, y de director del proceso.

Para más información sobre la implementación de este servlet, consulte el capítulo 10.

9.2.4. Takuan

Cuando el servlet ya ha recibido todos los ficheros necesarios y ha preparado el directorio para el proceso de negocio recibido, comienza a dirigir el proceso general de Takuan, enviando toda la información mostrada por las salidas estándares de vuelta al cliente.

El funcionamiento de Takuan ya se ha tratado en la parte I.

9.2.5. Diagrama

En la figura 9.2 se muestra la arquitectura conjunta del módulo de la interfaz, el cliente y el servlet.

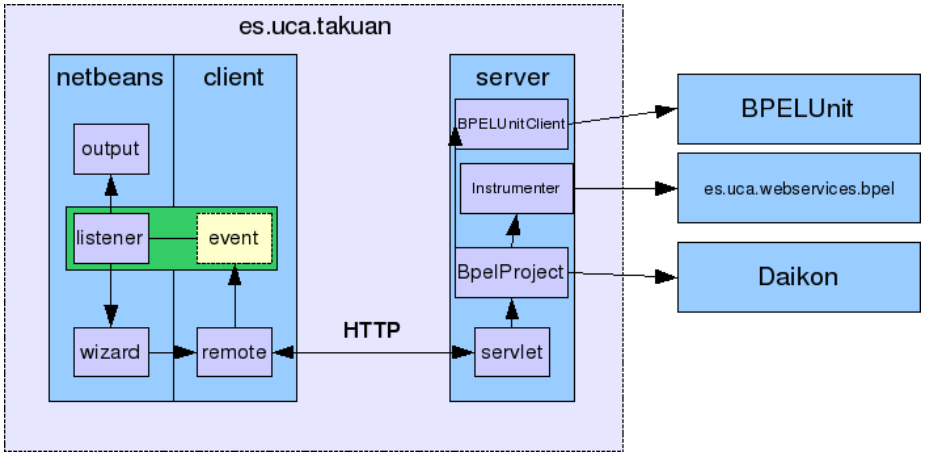

Figura 9.2: Diagrama de la arquitectura del plugin, servlet y partes de Takuan

9.3. Capturas

La figura 9.3 muestra una captura. Puede ver el resto de capturas detalladas del asistente en la sección 18.2.

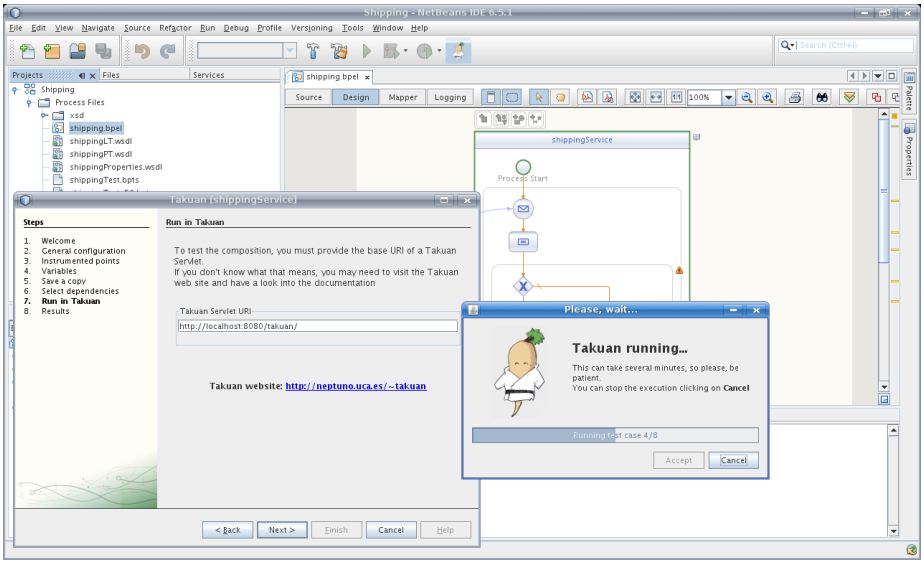

Figura 9.3: Captura de NetBeans con el plugin ejecutándose

9.4. Uso

En la parte III, capítulo 18, puede consultar el manual de instalación y uso de la interfaz gráfica desarrollada para usar Takuan desde el IDE NetBeans.

Capítulo 10

Servlet

Aunque en principio la interfaz gráfica podría comunicarse directamente con Takuan sin necesidad de ningún paso intermedio, se ha optado por seguir un esquema cliente-servidor fundamentalmente para permitir la distribución del programa en una máquina virtual.

El método recomendado de instalación de Takuan es dentro de una máquina virtual para evitar problemas con dependencias, y versiones no modificadas del software empleado por Takuan, como puede ser ActiveBPEL.

Se barajó inicialmente utilizar un protocolo propio para la comunicación, pero nos decantamos por implementar el servidor como un servlet por tres razones:

- El protocolo HTTP proporciona primitivas suficientes para las operaciones que se quiere realizar.
- La máquina virtual tiene configurado un servidor de aplicaciones - Tomcat 5.5 normalmente - sobre el que funciona ActiveBPEL. Esto facilita la instalación y configuración del servlet.
- HTTP es un protocolo sin estados, pero el servidor de aplicaciones sobre el que funcione el servlet se encarga de forma transparente de mantener las sesiones.

10.1. Clases

Todas las clases se encuentran dentro del módulo Java es.uca.takuan.server

BPELUnitClient Interactúa con BPELUnit a través de la API que éste suministra.

BpelProject Controla los pasos propios de Takuan. Es independiente del protocolo empleado para la comunicación.

InstrumenterTask Dirige las tareas de instrumentación: aplicación de hojas XSLT,...

Servlet Actúa de interfaz frente a las peticiones HTTP, las procesa y las pasa a la clase BpelProject.

La documentación generada por JavaDoc puede encontrarse en el CD adjunto.

10.2. Flujo de estados

El flujo de estados del servlet es mucho más sencillo, ya que las operaciones intermedias - como recibir o enviar fichero - no interfieren unas con otras. A continuación se describen estos estados junto a los eventos que desencadenan un cambio:

NEW El servlet está creado y listo para recibir solicitudes.

RECEIVING Se llega a este estado desde NEW, DONE o ERROR cuando se recibe la primera subida de un archivo a través del método HTTP `PUT`. Con el primer fichero, el servlet crea una instancia BpelProject que creará un directorio temporal para alojar los ficheros, y dirigirá todo el proceso de Takuan

PROCESSING Se llega a este estado desde RECEIVING con la primera solicitud `GET` al recurso *run*. A partir de este momento no se podrán seguir subiendo archivos. El proceso iniciado es:

1. Se lanza un nuevo hilo para el instrumentalizador, quien imprimirá la salida estándar de vuelta al cliente
2. Cuando finaliza el instrumentalizador, toma el control BPELUnitClient para interactuar con BPELUnit
3. Al finalizar la ejecución, BpelProject dirige la ejecución de los scripts de preprocesado y de ejecución de Daikon
4. Se cambia al estado DONE

DONE El proceso ha finalizado. Pueden recuperarse los ficheros con los invariantes (/invariants) e información de cobertura (/coverage) a través del método `GET`

ERROR Ha ocurrido un error durante alguna de las fases de ejecución

Capítulo 11

Conclusiones

11.1. Resumen

En esta segunda parte del documento se han presentado y descrito todos los trabajos de mejora realizados sobre el sistema de generación de invariantes para composiciones WS-BPEL Takuan. Fundamentalmente pueden resumirse en los siguientes puntos:

Recopilación de nuevos ejemplos Al ser un lenguaje nuevo, poco extendido y que implementa una lógica de negocio de empresa, la cantidad de ejemplos de uso reales son muy pocos. De hecho, la mayoría de artículos hasta hace poco trabajaban casi en exclusiva sobre el ejemplo del préstamo incluido en el estándar WS-BPEL.

Se han recopilado cinco ejemplos (seis, contando la variante síncrona de Shipping) que se ejecutan en Takuan.

Corrección de errores El sistema tenía algunos errores en el código de las hojas de transformación que dieron problemas al probar las nuevas composiciones. Esos errores han sido solucionados.

Mejora de soporte de WS-BPEL Se identificaron ciertas carencias en el soporte de características propias del estándar WS-BPEL, como, por ejemplo, las propiedades. Estas carencias han sido subsanadas.

Desarrollo de una interfaz gráfica Y de un servlet para poder conectar con una instalación de Takuan en otra máquina.

Obtención de información sobre cobertura De instrucciones, ramas y caminos. Estos datos permitirá a los usuarios mejorar sus casos de prueba, de forma que se generen invariantes más precisos.

Se han redactado tres artículos relacionados con el trabajo realizado sobre Takuan. Aparezco como autor principal en uno de ellos [Ál09], y como coautor en los otros dos [Pal09b, Gar09].

Además, y también como parte de este proyecto, se ha documentado el funcionamiento, el código y el uso de Takuan, dado que anteriormente la mayor parte de esta información estaba dispersa o, incluso, ausente.

Por último, parte de este trabajo ha servido para la redacción de tres artículos relacionados con Takuan, en los que también se ha colaborado como autor [Pal09b, Ál09, Gar09].

Capítulo 12

Trabajo futuro

12.1. Respecto a Takuan

Pese al cada vez mayor soporte del lenguaje WS-BPEL por parte de Takuan, siguen existiendo puntos del estándar que no están completamente admitidos, principalmente por la carencia de ejemplos que hagan uso de estas características. Por tanto, un punto prioritario en el trabajo futuro es seguir buscando, o diseñando, nuevos procesos que hagan un uso exhaustivo de todas las posibilidades del lenguaje. Una vez se tenga esa base, lo inmediatamente posterior es mejorar el soporte de estos nuevos elementos.

12.2. Respecto a la interfaz

En un futuro sería interesante lograr una integración mayor con el entorno de edición de las composiciones WS-BPEL, permitiendo al usuario marcar en la ventana de edición visual de la composición los elementos y variables que quiere instrumentar.

12.3. Respecto al servlet

Pueden identificarse tres puntos de mejora principalmente:

Seguridad Permitiendo autenticación de los usuarios antes de realizar las pruebas. Esto es algo que no se considera necesario en la actualidad, pero podría ser interesante implementar en un futuro.

Procesamiento por lotes De forma que no sea necesario esperar de forma síncrona a la finalización de la instrumentación y ejecución.

Servicio web estándar. Publicar información sobre las operaciones y las interfaces empleando WSDL, siendo así Takuan mismo un servicio web usable desde composiciones WS-BPEL.

12.4. Respecto a la documentación

Inevitablemente el trabajo sobre la documentación continuará mientras continúe el desarrollo de Takuan. Todas las nuevas características deberán documentarse.

12.5. Soporte de BPELscript

Como ya se ha dicho varias veces con anterioridad, WS-BPEL es un lenguaje basado en XML. Sin embargo, los programadores no suelen escribir código en XML, al resultar una sintaxis poco cómoda. El grupo de Apache ODE (*Orchestration Director Engine*) realizó varias propuestas de lenguajes de script traducibles a WS-BPEL y viceversa para facilitar la programación y mantenimiento. BPELscript se basa en este trabajo y proporciona, según su autor [Bis]:

- Una sintaxis compacta inspirada en lenguajes de scripting como JavaScript y Ruby
- Cobertura completa de todas las características de BPEL
- Traducción desde WS-BPEL 2.0
- Traducción a WS-BPEL 2.0

Es una tecnología reciente, pero que va ganando popularidad. Por tanto, lograr que Takuan admita como entrada procesos escritos en BPELscript es una meta que se quiere lograr.

Capítulo 13

Agradecimientos

- A Manuel Palomo Duarte, por darme la oportunidad de colaborar en este proyecto.
- A Antonio García Domínguez, por su excelente trabajo de documentación y toda la ayuda prestada.
- A Juan José Domínguez, Antonia Estero, Inmaculada Medina y Francisco Palomo del grupo de investigación *SPI&FM*, por su apoyo.

Parte III

Manual de uso de Takuan

Capítulo 14

Obtener Takuan

Como ya se ha descrito con anterioridad, Takuan no es una única aplicación, sino un conjunto de programas y utilidades que trabajan juntas. Es por ello que la instalación no es tan trivial como descargar y ejecutar un instalador, o un paquete para una distribución de un sistema operativo.

La instalación de Takuan requiere de la descarga, instalación y parcheado de sus componentes, además de su configuración. Por ello se recomienda realizar la instalación en una máquina virtual, o en una copia virtual separada del sistema (usando chroot).

14.1. Instalación desde cero

Se explicará la instalación de Takuan [Dom09] sobre un sistema Ubuntu Server 8.04 LTS.

14.1.1. Instalación de Tomcat

Tanto ActiveBPEL como el servlet de Takuan funcionan bajo Tomcat, por lo que lo primero que necesitaremos instalar es este servidor de aplicaciones. Concretamente, emplearemos la versión 5.5 disponible en el repositorio de Takuan:

```
$ svn export
  http://neptuno.uca.es/svn/sources-fm/deps/apache-tomcat-5.5.26.zip
```

Tras la descarga, descomprimiremos el zip en `/opt`.

14.1.2. Instalación de ActiveBPEL 4.1

El primer paso antes de empezar la instalación de ActiveBPEL es configurar la variable de entorno CATALINA_HOME. Para ello editaremos el fichero /etc/environment y añadiremos la línea:

```
CATALINA_HOME="/opt/tomcat5.5"
```

La ruta mostrada es la creada por el paquete de Tomcat instalado. Para otras versiones u otros sistemas puede diferir.

Lo siguiente será instalar el script encargado de lanzar ActiveBPEL. Para ello, lo descargaremos del repositorio en `/usr/bin`.

```
$ sudo svn export
  http://neptuno.uca.es/svn/sources-fm/scripts/ActiveBPEL.sh
```

El siguiente paso es instalar algunas dependencias que necesitaremos para poder compilar la versión modificada de ActiveBPEL:

```
$ sudo aptitude install curl patch ant openjdk-6-jdk
```

Tras lo cual descargaremos el código de ActiveBPEL, lo parchearemos, compilaremos e instalaremos:

```
$ cd /tmp
$ svn export
  http://neptuno.uca.es/svn/sources-fm/deps/activebpel-4.1-src.tar.gz
$ svn export
  http://neptuno.uca.es/svn/sources-fm/src/parches/ActiveBPEL_delta_4.1.diff.gz
$ tar xzf activebpel-4.1-src.tar.gz
$ cd activebpel-4.1
$ zcat ../ActiveBPEL_delta_4.1.diff.gz | patch -p1
$ bash recompile.sh
```

Si todo va bien, podremos acceder a `http://localhost:8080/BpelAdmin` en nuestro servidor. Si no funcionara, debería bastar con matar manualmente todos los procesos de Java y lanzar de nuevo el servidor con la orden "ActiveBPEL.sh start".

14.1.3. Instalación de BPELUnit

Es posible hacer la instalación de dos formas: descargando el código, aplicando el parche y compilando; o directamente descargar una versión precompilada.

Por comodidad, trabajaremos con la versión ya compilada.

Lo primero es descargar el archivo:

```
$ svn export
  http://neptuno.uca.es/svn/sources-fm/deps/bpelunit-CVS-200709+parches.tar.bz2
```

Y lo descomprimimos en `/opt`. Lo siguiente será modificar el fichero de configuración `/opt/bpelunit/conf/configuration.xml` para que los valores sean los correctos. En principio, sólo habrá que modificar el valor de ActiveBPELDeploymentDirectory y darle el valor `/opt/tomcat5.5/bpr`.

Por último, damos los permisos adecuados al directorio `/opt/tomcat5.5/bpr`

```
$ chmod 1777 /opt/tomcat5.5/bpr
```

y añadimos otra variable de entorno más a `/etc/environment`

```
BPELUNIT_HOME="/opt/bpelunit"
```

14.2. Entorno de instrumentación

Para la instrumentación necesitamos instalar Saxon-B, un motor XSLT para Java.

```
$ svn export http://neptuno.uca.es/svn/sources-fm/deps/saxonb9-1-0-1j.zip
$ unzip saxonb9-1-0-1j.zip
$ sudo mv saxonb9.jar saxon9-dom.jar /usr/share/java
```

Tras lo que modificaremos el fichero .bashrc de nuestro usuario para añadir al classpath los dos nuevos ficheros jar:

```
$ echo 'export SAXON_HOME="/usr/share/java"' >> ~/.bashrc
$ echo 'export CLASSPATH="$CLASSPATH:/usr/share/java/saxonb9.jar:
       /usr/share/java/saxon9-dom.jar"' >> ~/.bashrc
```

También pueden añadirse al fichero `/etc/environment` si queremos que todos los usuarios compartan esas variables de entorno.

Ahora podemos aplicar los cambios cerrando y abriendo sesión, o ejecutando el script `.bashrc` en el ambiente actual.

```
$ source ~/.bashrc
```

Para comprobar si hemos realizado los pasos correctamente, podemos probar a ejecutar:

```
$ java net.sf.saxon.Transform
```

14.3. Entorno de análisis

14.3.1. Módulos para Perl

Para el análisis debemos instalar los módulos para Perl Graph::UnionFind y Parallel:ForkManager

```
$ sudo aptitude install libgraph-perl libparallel-forkmanager-perl
```

14.3.2. Modula 3

También es necesario compilar e instalar Modula 3. Para ello, primero debemos instalar las dependencias de compilación:

```
$ sudo aptitude install gcc binutils flex freeglut3-dev\
                        unixodbc-dev libmotif-dev\
                        libx11-dev libxmu-dev libpq-dev
```

Lo siguiente será obtener cm3-min del repositorio e instalarlo.

```
$ mkdir cm3-min
$ cd cm3-min
$ svn export http://neptuno.uca.es/svn/sources-fm/deps/
  cm3-min-POSIX-LINUXLIBC6-d5.6.0-2008-02-23-03-35-01.tgz
```

```
$ tar xzf cm3-min-*
$ sudo mkdir /usr/local/cm3
$ sudo chmod 0777 /usr/local/cm3
$ ./cminstall
```

Añadimos las bibliotecas de CM3 a la lista de búsqueda global, y los binarios al PATH:

```
$ echo "/usr/local/cm3/lib" | sudo tee /etc/ld.so.conf.d/cm3.conf
$ echo 'export PATH=$PATH:/usr/local/cm3/bin' >> ~/.bashrc
```

Salimos y volvemos a entrar, y probamos la configuración con

```
$ cm3 -version
```

Ahora debemos descargar el código

```
$ mkdir ~/cm3-src
$ cd ~/cm3-src
$ svn export http://neptuno.uca.es/svn/sources-fm/deps/
  cm3-src-all-d5.6.0-2008-03-04-01-01-39.tgz
$ tar xzf cm3-src-*
$ cd scripts/
$ ./do-cm3-core.sh buildship && \
    ./install-cm3-compiler.sh upgrade && \
    ./do-cm3-std.sh buildship && \
    ./do-cm3-std.sh buildship
```

14.3.3. Simplify

Simplify es un demostrador de teoremas que necesita tener completamente instalado Modula 3.

El primer paso, como es habitual, es descargar el código del repositorio:

```
$ mkdir simplify
$ cd simplify
$ svn export http://neptuno.uca.es/svn/sources-fm/deps/
  simplify-SVN71394.tar.bz2
$ tar xjf simplify-SVN71394.tar.bz2
$ cd simplify/Simplify
$ for i in cgi-utils digraph prover simplify;\
 do pushd $i && cm3 && cm3 -ship && popd; done
```

Si ejecutamos ahora Simplify debería aparecer un *prompt* vacío que podemos cerrar con la combinación de teclas *Ctrl+C*. Si funciona, podemos restringir los permisos de escritura bajo `/usr/local/cm3`.

```
$ sudo chown -R root.root /usr/local/cm3
$ sudo find /usr/local/cm3 -execdir chmod go-w '{}' \;
```

14.4. Instalación de Daikon

Lo primero será instalar los paquetes necesarios para compilar programas en C, dado que el código Java de Daikon emplea el preprocesador de C en algunos sitios.

```
$ sudo aptitude install build-essential
```

Obtenemos el código de Daikon, lo descomprimimos, aplicamos el parche y compilamos.

Antes de comenzar, debemos asegurarnos de que está instalado sun-java6-jdk.

```
$ cd /tmp
$ svn export
  http://neptuno.uca.es/svn/sources-fm/deps/daikon-4.3.4.tar.gz
$ svn export
  http://neptuno.uca.es/svn/sources-fm/src/parches/Daikon_delta_4.3.4.diff.gz
$ cd /opt
$ sudo su -
$ tar xzf /tmp/daikon-4.3.4.tar.gz
$ chown root.root -R daikon
$ cd daikon
$ zcat /tmp/Daikon_delta_4.3.4.diff.gz | sudo patch -p1
$ cd java
$ export DAIKONBIN=/opt/daikon/bin/
$ export CLASSPATH=/usr/lib/jvm/java-6-sun-1.6.0.16/lib/tools.jar
$ make daikon
```

Por último, añadiremos el guión `deps/manual/guiones/daikon.sh` y el fichero de configuración `deps/manual/guiones/daikon.options` a `/usr/local/bin`. Tendremos que modificar DAIKON y DAIKON_OPTS_FILE para que apunten a las rutas correctas.

```
$ svn export
  http://neptuno.uca.es/svn/sources-fm/scripts/daikon.sh
$ svn export
  http://neptuno.uca.es/svn/sources-fm/scripts/daikon.options
$ sudo cp /tmp/daikon.{options,sh} /opt/daikon
$ sudo ln -s /opt/daikon/daikon.sh /usr/local/bin
```

14.5. Instalación de Takuan

El último paso será instalar el framework Takuan. Para ello, deberemos descargarnos el fichero con la distribución de Takuan del repositorio y descomprimirlo donde se desee.

```
$ svn export http://neptuno.uca.es/svn/sources-fm/bin/takuan-dist.zip
$ unzip takuan-dist.zip
```

La distribución incluye ejemplos y guiones Ant para probar que la instalación se ha realizado correctamente.

Por último, se debe añadir una línea a `/.bashrc` declarando la variable de entorno TAKUAN_HOME, con la ruta al directorio en que se halla Takuan, sin la barra final.

```
$echo 'export TAKUAN_HOME=/home/usuario/takuan' >> ~/.bashrc
```

De nuevo, si queremos compartir la instalación, descomprimiremos el fichero en un directorio compartido (`/opt`) y la variable de entorno TAKUAN_HOME la añadiremos al fichero `/etc/environment`.

14.6. Servlet de Takuan

Para poder instalar el servlet de Takuan, de forma que podamos usarlo de forma remota - concretamente, desde el asistente para NetBeans - necesitamos tener instalados todos los elementos anteriores.

Las dependencias para su compilación son:

- Tener instalado libservlet2.4-java
- Tener configuradas las variables de entorno BPELUNIT_HOME, TAKUAN_HOME, SAXON_HOME y CATALINA_HOME

Una vez nos aseguremos de tener las dependencias cubiertas, descargamos el código fuente:

```
$ svn export http://neptuno.uca.es/svn/sources-fm/src/Takuan_Server
  TakuanServer
```

Lo compilamos y reiniciamos el servicio:

```
$ cd TakuanServer
$ ant -f standalone-build.xml clean local-deploy
$ ActiveBPEL.sh restart
```

Ahora, bajo `http://localhost:8180/takuan/` tendremos disponible el servicio.

14.7. Máquina virtual

Para facilitar a los usuarios el uso de la herramienta Takuan, se distribuye una imagen de una máquina virtual de VirtualBox [Micc] instalada y configurada. Por lo general, se recomienda emplear este método.

Los pasos a seguir son tan simples como:

1. Descargar e instalar VirtualBox
2. Descargar la imagen virtual del sitio web de Takuan [1]

[1] `http://neptuno.uca.es/redmine/wiki/takuan-website/Takuan_Virtual_Machine`

3. Registrar la imagen de disco en VirtualBox

4. Crear una nueva máquina virtual con la imagen de disco de Takuan. Se recomienda emplear al menos 512 MB de RAM para la máquina virtual, aunque con sólo 256 puede funcionar.

Opcionalmente, puede emplearse como referencia la configuración disponible en el sitio web de Takuan para permitir el acceso a la máquina virtual a través de SSH y HTTP - lo que será útil si quiere conectarse al servlet de Takuan.

Capítulo 15

Preparar la composición

Aunque Takuan puede trabajar sobre una composición WS-BPEL sin modificaciones, existen extensiones propias que permiten configurar los parámetros de ejecución de Takuan, y especificar qué variables y puntos de programa se quieren inspeccionar.

Para hacer uso de estas extensiones, primero es necesario definir el espacio de nombres de Takuan en el documento. Además, si se quiere que ActiveBPEL pueda ejecutar directamente la composición modificada, se debe añadir un bloque que declare que el motor de ejecución puede ignorar las extensiones propias de XPath que usa.

Listing 15.1: Declaración del espacio de nombres de Takuan

```
1 <bpel:process
2    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/
         process/executable"
3    xmlns:uca="http://www.uca.es/xpath/2007/11">
4     <bpel:extensions>
5         <bpel:extension mustUnderstand="no"
6            namespace="http://www.uca.es/xpath/2007/11"/>
7     </bpel:extensions>
8 </bpel:process>
```

Con esta base, ya podemos especificar los parámetros de instrumentalización e inspección.

15.1. Parámetros de la configuración

Hay tres parámetros generales personalizables:

uca:instrumentVariablesByDefault Acepta los valores "yes" y "no". Especifica si se instrumentarán todas las variables por defecto o no.

uca:instrumentSequencesByDefault Acepta los valores "yes" y "no". Especifica si se instrumentan todas las secuencias por defecto, o sólo las marcadas.

uca:maxInstrumentationDepth Acepta valores enteros mayores o iguales a 0. Especifica la profundidad máxima del árbol XML que se instrumentará. 0 significa sin límite.

Ambos son atributos del elemento raíz <process> de una composición WS-BPEL.

15.2. Variables

Para las variables hay un único atributo: `uca:inspeccionar`. Especifica si se debe o no inspeccionar la variable. Los valores posibles son:

yes Si se inspeccionará

no No se inspeccionará

sin definir Se usará el comportamiento configurado por el atributo uca:instrumentVariablesByDefault del elemento raíz

A continuación se muestra un ejemplo con los tres posibles valores:

```
1  <variables>
2    <variable name="shipRequest"
3          messageType="sif:shippingRequestMsg"
4          uca:inspeccionar="yes"/>
5    <variable name="shipNotice"
6          messageType="sif:shippingNoticeMsg"
7          uca:inspeccionar="no"/>
8    <variable name="itemsShipped"
9          type="ship:itemCountType"/>
10 </variables>
```

15.3. Puntos de programa

Para cada elemento <sequence> del programa, puede especificarse si se quiere instrumentar o no. Esto se realiza por el atributo `uca:instrument`, cuyo uso es similar al equivalente para variables.

yes Instrumentar la secuencia, incluso aunque su profundidad sea mayor que el límite configurado, de haberlo

no No instrumentar la secuencia

no definido Se considerará el comportamiento configurado por el atributo `uca:instrumentSequencesByDefault` del elemento raíz. Si el valor por defecto es "yes", pero la profundidad es mayor que el límite especificado por `uca:maxInstrumentationDepth`, **no** se instrumentará

Capítulo 16

Casos de prueba

Una vez que tenemos el fichero WS-BPEL adaptado, debemos especificar un conjunto de casos de prueba que deseamos ejecutar. Aquí se realiza una breve explicación de cómo realizarlo, pero si desea mayor nivel de detalle, puede consultarlo en la documentación de BPELUnit [May06b].

16.1. Elemento raíz y espacio de nombres

Los ficheros con extensión `bpts` son ficheros XML. Como todos los documentos XML, incluye un elemento raíz donde se especifican los espacios de nombre:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <tes:testSuite
3     xmlns:trs="http://www.sun.com/javaone/05/
          TravelReservationService"
4     xmlns:ota="http://www.opentravel.org/OTA/2003/05"
5     xmlns:tes="http://www.bpelunit.org/schema/testSuite"
6     xmlns:ves="http://www.sun.com/javaone/05/
          VehicleReservationService"
7     xmlns:aes="http://www.sun.com/javaone/05/
          AirlineReservationService"
8     xmlns:hes="http://www.sun.com/javaone/05/
          HotelReservationService">
9 </tes:testSuite>
```

En este caso, el prefijo `tes` está asociado al espacio de nombres de BPELUnit. Los demás son prefijos propios de esta composición. Serán necesarios más adelante para definir los puertos, los tipos, etc.

16.2. Información sobre el proceso

El primer bloque de información incluirá datos sobre el proceso que será desplegado: nombre, URL, partners - servicios a los que se conecta -, etc.

```
1 <tes:name>TravelReservationService</tes:name>
2 <tes:baseURL>http://localhost:7777/ws</tes:baseURL>
```

```
3
4  <tes:deployment>
5    <tes:put name="TravelReservationService" type="
         activebpel">
6      <tes:wsdl>TravelReservationService.wsdl</tes:wsdl>
7      <tes:property name="BPRFile">travel.bpr</tes:property
           >
8    </tes:put>
9    <tes:partner name="Airline" wsdl="
         AirlineReservationService.wsdl"/>
10   <tes:partner name="Hotel"    wsdl="
         HotelReservationService.wsdl"/>
11   <tes:partner name="Vehicle" wsdl="
         VehicleReservationService.wsdl"/>
12 </tes:deployment>
```

tes:name Un nombre identificativo para el proceso

tes:baseURL URL sobre la que se desplegará. Normalmente, el valor del ejemplo es el que siempre se deberá usar

test:deployment Información sobre el despliegue del proceso

tes:put Proceso a desplegar. Se especifica su nombre, el motor que lo ejecutará, el fichero WSDL que describe el servicio, y el fichero `bpr` que contiene la composición

tes:partner Servicios externos a los que se conectará, junto a su descripción WSDL. Estos servicios externos pueden sustituirse por *mockups* cuando se especifique el caso de prueba. El atributo `name` debe identificar unívocamente al partner, ya que será el nombre que se usará después para sustituirlo por uno simulado.

16.3. Casos de prueba

El último bloque son los casos de prueba. Se especifican como un elemento raíz <tes:testCases> en el que se anidan elementos <tes:testCase> para cada caso de prueba.

Un ejemplo de un caso de prueba sería:

```
1  <tes:testCase name="OnlyAirline" basedOn="" abstract="
       false" vary="false">
2    <tes:clientTrack>
3      <tes:sendReceive
4           service="trs:TravelReservationSoapService"
5           port="TravelReservationSoapHttpPort"
6           operation="buildItinerary">
7
8        <tes:send fault="false">
9          <tes:data>
10           <ota:TravelItinerary/>
```

```
11          </tes:data>
12        </tes:send>
13
14        <tes:receive fault="false">
15          <tes:condition>
16            <tes:expression>ota:...</tes:expression>
17            <tes:value>'M839LWAA'</tes:value>
18          </tes:condition>
19        </tes:receive>
20
21      </tes:sendReceive>
22    </tes:clientTrack>
23
24    <tes:partnerTrack name="Airline">
25      <tes:receiveSendAsynchronous>
26        <tes:receive service="
              aes:AirlineReservationSoapService"
27                     port="AirlineReservationSoapHttpPort"
28                     operation="reserveAirline"
29                     fault="false"/>
30
31        <tes:send service="
              aes:AirlineReservationCallbackService"
32                  port="
                        AirlineReservationCallbackSoapHttpPort
                        "
33                  operation="airlineReserved"
34                  fault="false">
35          <tes:data>
36            <ota:TravelItinerary/>
37            </ota:TravelItinerary>
38          </tes:data>
39        </tes:send>
40      </tes:receiveSendAsynchronous>
41    </tes:partnerTrack>
42
43    <tes:partnerTrack name="Vehicle">
44      <!-- No es invocado -->
45    </tes:partnerTrack>
46
47    <tes:partnerTrack name="Hotel">
48      <!-- No es invocado -->
49    </tes:partnerTrack>
50
51  </tes:testCase>
```

Se ha obviado el contenido de los mensajes por claridad y ahorro. Puede consultar este ejemplo y otros en el CD adjunto.

A continuación se describen los elementos que aparecen:

tes:testCase Como ya se ha dicho con anterioridad, elemento para definir un caso de prueba. El atributo `name` es un identificador único del caso de prueba, Un caso de prueba puede basarse en uno anterior, definido con el atributo `basedOn`. Si el atributo `abstract` está a "yes", entonces el caso de prueba no puede ser ejecutado. Por último, `vary` especifica si se debe ejecutar más de una vez el mismo caso de prueba cambiando tiempos de respuesta, etc.

tes:clientTrack Define la petición que se enviará desde el cliente simulado.

tes:sendReceive El cliente del proceso mandará una petición síncrona. Los atributos especifican a qué servicio y puerto se enviará la petición, y qué operación se usará.

tes:send Los datos incluidos dentro de tes:data se enviarán al proceso. Estos datos seguirán la estructura definida por el propio proceso (de ahí el usar el prefijo `ota`). El atributo `fault` a "false" indica que no debe simularse un fallo.

tes:receive Se evaluará cuando el proceso responda al cliente. Con `tes:condition` pueden comprobarse si ciertas condiciones esperadas en el mensaje de respuesta se cumplen.

tes:partnerTrack Aquí se define un mockup. La estructura es prácticamente idéntica al `tes:clientTrack`. El atributo `name` define qué servicio externo está siendo reemplazado.

tes:receiveSendAsynchronous Se reciben primero los datos, y se responde de forma asíncrona (el proceso cliente no espera la respuesta).

La estructura anterior se repetirá tantas veces como casos de prueba se desee.

Capítulo 17

Uso y ejecución

El ciclo de uso habitual de Takuan viene reflejado en el diagrama 17.1.

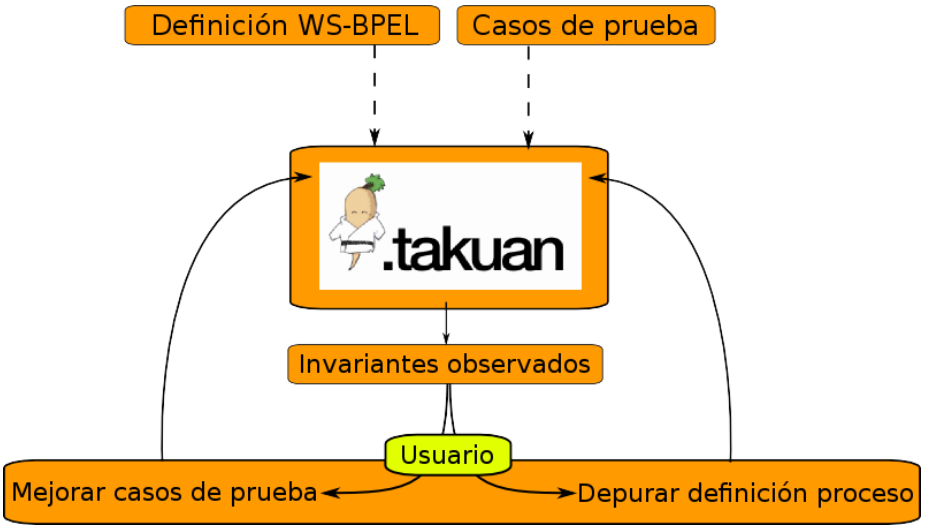

Figura 17.1: Diagrama de uso de Takuan

Takuan ejecuta un conjunto de casos de prueba original en un proceso WS-BPEL. A partir de los invariantes obtenidos, y de la información de cobertura, puede ser que los invariantes que obtengamos sean los esperados o que hay alguno que consideremos incorrecto. En este segundo caso se puede optar por mejorar el conjunto de casos de prueba o depurar, si es necesario, el proceso y volver a ejecutar Takuan para comprobar la mejora.

17.1. Ejecución con guiones Ant

Takuan proporciona una tarea que puede ser usada desde un guión Ant, pero no proporciona ninguna interfaz de línea de comandos. Por tanto, la única forma de ejecutar Takuan es mediante guiones Ant - se incluyen ejemplos en el CD - o mediante el asistente para NetBeans descrito en el capítulo 18.

En caso de duda o desconocimiento del lenguaje empleado por Ant, puede consultar su manual oficial [Apa].

Lo primero que deberemos tener en el guión es la definición de la nueva tarea:

```
1 <taskdef name="instrument"
2       classname="es.uca.webservices.bpel.
            InstrumentBPELProcessTask"
3       classpathref="instr.classpath"/>
```

Y lo siguiente, será definir una nueva tarea para BPELUnit. Observe que se espera que la variable de entorno BPELUNIT_HOME contenga el valor del directorio de instalación de BPELUnit.

```
1 <typedef name="bpelunit" classname="org.bpelunit.
      framework.ui.ant.BPELUnit">
2   <classpath>
3     <fileset dir="${env.BPELUNIT_HOME}/lib"/>
4   </classpath>
5 </typedef>
```

De esta forma podemos emplear la clase que dirigirá el proceso como una tarea más de Ant.

Lo siguiente será definir las tareas que se deben realizar, pero para ello primero es útil conocer el orden en el que se deben ejecutar, de forma que se sepan las dependencias:

Instrumentalización La tarea que acabamos de definir guiará el proceso de instrumentalización. Cuando finalice, tendremos una serie de ficheros intermedios y un archivo comprimido con extensión `bpr` que contendrá todo lo necesario para realizar la ejecución

Ejecución de BPELUnit Se enviará a BPELUnit el fichero `bpts` donde se especifica el conjunto de casos de prueba, y el fichero `bpr` sobre el que se ejecutará las pruebas

Preparación de las trazas Se llamará al script que inicia el análisis de las trazas de ejecución, transformándolas al formato entendido por Daikon y obteniendo la información sobre la cobertura. Por último, llamará a Daikon para obtener los invariantes

En los ejemplos contenidos en el CD, estos pasos, que son comunes a todos los procesos WS-BPEL, están definidos en un fichero llamado `base-build.xml`, incluido por los scripts Ant propios de cada composición, en los que se especifica el nombre de la composición, la salida, etc.

Como se incluye en el CD, se omite el código por claridad. La descripción dada ayudará a la comprensión del guión.

17.2. Descripción de la salida

Los ficheros generados por Takuan con información relevante para el usuario son cinco:

coverage.log Contiene información sobre la cobertura de sentencias y de ramas

pathCoverageAll.log Contiene todos los posibles caminos de la composición

pathCoverageExecuted.log Contiene los caminos que se han recorrido en algún momento por alguno de los casos de prueba

pathCoverageNotExecuted.log Contiene los caminos que nunca se han recorrido

procesoInspeccionado.out Contiene la lista de invariantes inferidos para cada punto de programa

Si ejecuta Takuan empleando un script Ant, encontrará estos ficheros bajo el directorio que haya configurado en el guión. Por defecto, este directorio de destino tendrá la forma daikon-out-<fecha>-<hora>.

Si ejecuta Takuan a través de la interfaz para NetBeans - o, lo que es lo mismo, a través de su interfaz web - se descargarán remotamente.

NOTA: La información de cobertura se puede generar tanto en modo texto plano (*human readable*) como en XML. Aquí sólo se menciona el formato legible por humanos.

17.2.1. Cobertura de sentencias

El fichero `coverage.log` tiene dos partes. La primera, encabezada por las siguientes dos líneas:

```
Instruction coverage
====================
```

es la que contiene la información sobre la cobertura de sentencias. A su vez, está separada en dos bloques: sentencias ejecutadas y sentencias nunca ejecutadas.

Se muestra un ejemplo de cada caso:

```
Executed 45/50 (90%)
--------------------
(1/1) /process/sequence/if/else/sequence/while/sequence/assign/copy[0]
(1/1) /process/sequence/assign[3]
(1/1) /process/sequence/reply[@name='Respuesta']
(1/1) /process/sequence
```

En este fragmento podemos ver la cobertura de sentencias que tenemos: un 90 %. Hay un 10 % de instrucciones (5 sobre el total) que nunca se han ejecutado.

Bajo la cabecera, se muestra la lista de sentencias ejecutadas, precedidas de una fracción del tipo (número de veces ejecutadas/número de casos de prueba), dado que no es suficiente una cobertura del 100 % para obtener buenos resultados. También es necesario ejecutarlas un número suficiente de veces.

```
Never executed 5/50 (10%)
-------------------------
/process/sequence/if/if-condition/sequence/assign[2]
/process/sequence/if/if-condition/sequence
/process/sequence/if/if-condition/sequence/assign[3]
```

Aquí se muestra parte del bloque de sentencias nunca ejecutadas. Lógicamente, es el conjunto complementario al de ejecutadas.

17.2.2. Cobertura de ramas

En la segunda mitad del fichero coverage.log muestra la cobertura de ramas. En este caso, se encabeza por las líneas:

```
Branches coverage
=================
```

El formato es similar al anterior, sólo que únicamente se muestran las ramas

```
Branches executed 1 / 2 (50%)
-----------------------------
/process/sequence/if/else

Branches not executed 1 / 2 (50%)
---------------------------------
/process/sequence/if/if-condition
```

En este ejemplo tenemos una cobertura del 50 % sobre dos ramas. Es decir, hay una rama, concretamente la rama del if, que nunca se ha ejecutado. Sería necesario ampliar el conjunto de casos de prueba.

17.2.3. Cobertura de caminos

La información de la cobertura de caminos dado su volumen, se encuentra separado en tres ficheros. Generalmente el que más nos interesa es el que contiene los caminos nunca ejecutados: `pathCoverageNotExecuted.log`

En cualquier caso, el formato de los tres es el siguiente: estadísticas generales, camino 1, camino 2, ...

Por ejemplo, en el caso de todos los caminos posibles:

```
Maximum length of an execution path: : 46
Possible execution paths: 2
Minimum length of an execution path: : 36
```

Hay dos caminos de ejecución posible, siendo el número máximo de instrucciones posible 46, y el menor, 36.

En el caso de caminos no ejecutados:

```
Number of not executed paths: 1
```

Hay un camino que no se ha ejecutado.

En todos los casos, estas estadísticas vienen seguidas de una lista de caminos posibles. Por ejemplo, extraído de un ejemplo de no ejecutados:

```
Execution path no.1
36 sentences

/process
/process/sequence
/process/sequence/receive
/process/sequence/assign
/process/sequence/assign/copy[0]
/process/sequence/assign/copy[1]
/process/sequence/assign/copy[2]
/process/sequence/assign/copy[3]
/process/sequence/assign/copy[4]
/process/sequence/assign[2]
/process/sequence/assign[2]/copy[0]
/process/sequence/if
/process/sequence/if/if-condition
/process/sequence/if/if-condition/sequence
```

El camino número uno está compuesto por 36 sentencias. Y, como podemos comprobar en este caso en concreto, el camino no cubierto es la rama del if del primer condicional - lo que coincide con lo que obtuvimos con la cobertura de ramas.

Recordamos que una cobertura del 100 % de caminos, implica una cobertura del 100 % de ramas, y del 100 % de sentencias, pero lo contrario no es necesariamente cierto.

Capítulo 18

Plugin para NetBeans

El plugin para NetBeans permite usar Takuan de forma cómoda y fácil, a través de un asistente gráfico que se integra en el IDE NetBeans [Mica]. Este plugin, no obstante, sólo permite preparar el fichero BPEL y mandarlo, junto a sus dependencias, al sistema Takuan.

Esta comunicación con Takuan se hace a través de un Servlet que recibe el proceso, y dirige su ejecución y el procesado de su salida. Es decir, la interfaz gráfica - plugin - y el sistema Takuan pueden estar en máquinas separadas. Esto permite tener Takuan dentro de una máquina virtual, y comunicar con él desde el sistema anfitrión.

Si no dispone de una máquina - virtual o no - con Takuan y el servlet instalados, consulte el capítulo 14 para ver una guía de instalación.

18.1. Instalación

Para poder hacer uso del plugin para NetBeans, primero necesita tener instalado un entorno NetBeans con soporte para WS-BPEL.

Suponiendo que disponga de una instalación con esas características, lo siguiente sería descargar el paquete con el plugin:

`http://neptuno.uca.es/files/netbeans/es-uca-takuan.nbm`

Una vez disponga del paquete, abra NetBeans y vaya al menú *Tools / Plugins*. Seleccione la pestaña *Downloaded* y haga click en *Add plugins*. En el diálogo de selección de fichero seleccione el fichero NBM que se ha descargado. Puede ver un ejemplo en la figura

Después seleccione el paquete "Takuan_NetBeans" y haga click en *Install*.

Se iniciará el asistente de importación de NetBeans. Simplemente tiene que seguirlo.

18.2. Uso

A continuación se describe el uso siguiendo el ciclo habitual de la aplicación.

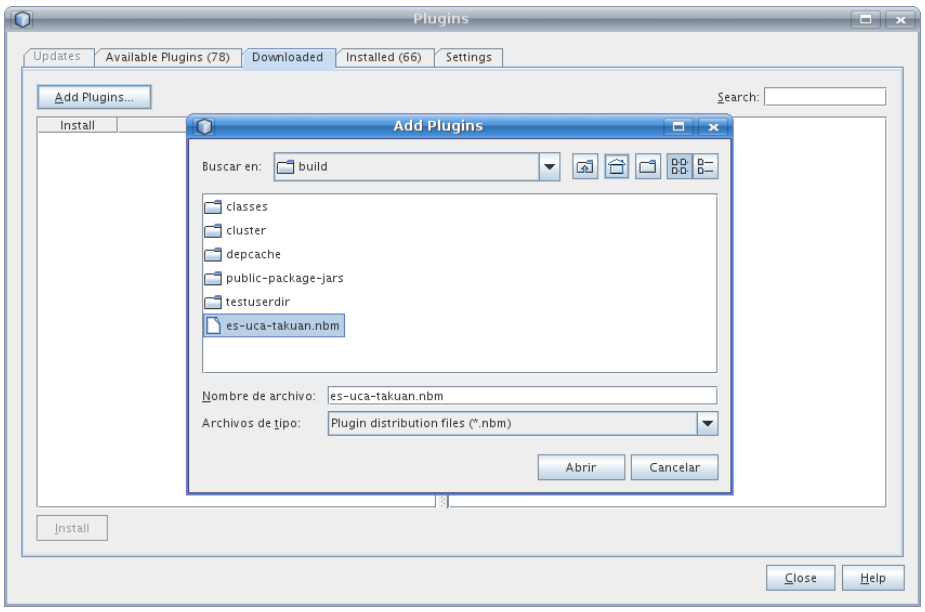

Figura 18.1: Selección del paquete

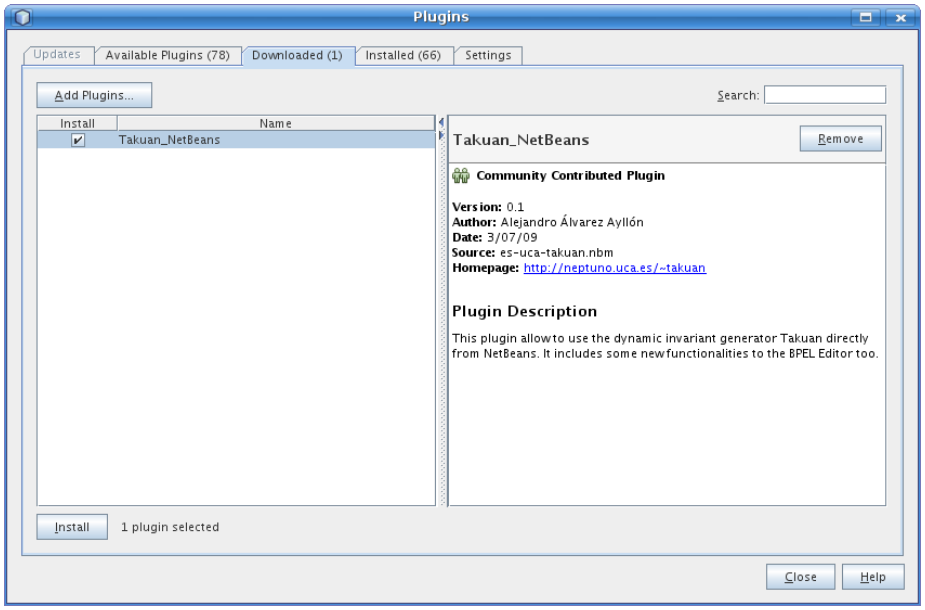

Figura 18.2: Selección del plugin

18.2.1. Creación / importación

Puede crear un proyecto desde cero siguiendo la guía *Developer Guide to BPEL Designer* [May06a], o bien crear un proyecto vacío y copiar todos los ficheros de una composición ya existente - incluyendo ficheros WSDL, XSD,... - a la carpeta `src` situada dentro del directorio del proyecto.

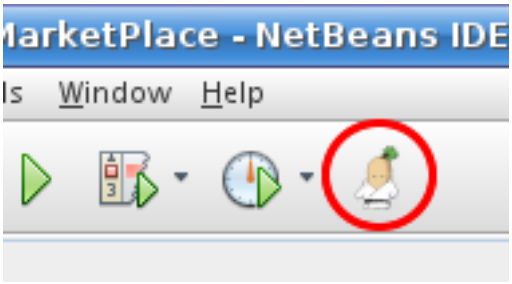

Figura 18.3: Inicio del asistente

18.2.2. Adaptación de la composición WS-BPEL

Se realiza de la forma descrita en el capítulo 15. Puede emplear el editor de código WS-BPEL que incluye NetBeans.

18.2.3. Asistente para Takuan

Para poder ejecutar el asistente de Takuan, debe

1. Tener el proyecto BPEL marcado como *Main Project*
2. Tener el fichero BPEL seleccionado o activo en el editor

Y hacer click sobre el botón representado en la figura 18.3.
Entonces aparecerá la ventana de bienvenida al asistente.

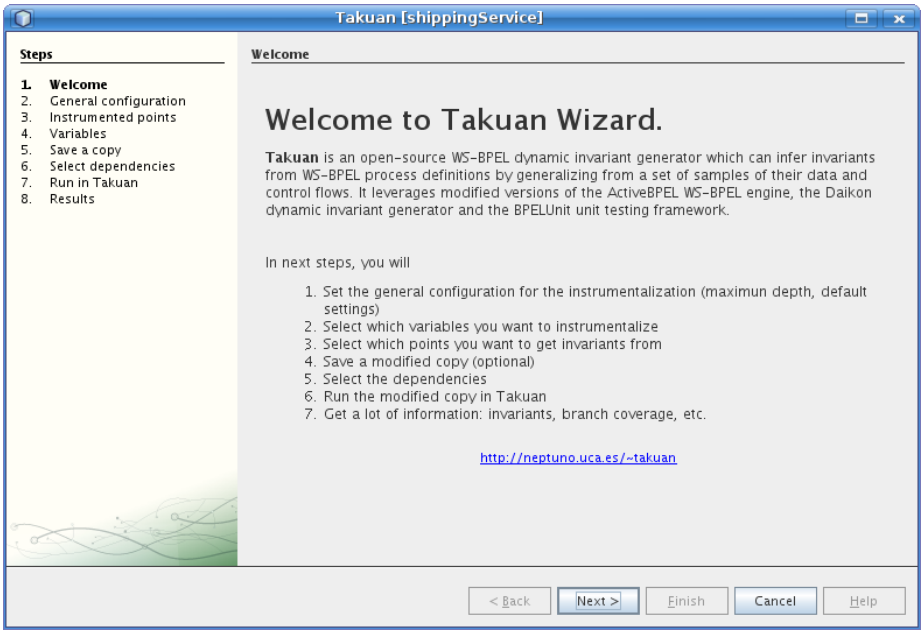

Figura 18.4: Detalle de la ventana de bienvenida al asistente

18.2.4. Parámetros de la ejecución

En el primer paso se le solicitará que configure dos parámetros relacionados con la ejecución e instrumentación:

Instrument variables by default Si se marca, todas las variables se instrumentarán salvo que se indique lo contrario. Si se desmarca, sólo se instrumentarán las especificadas explícitamente

Maximum depth Profundidad máxima del árbol XML que se instrumentará

Coverage output format Formato de salida de los datos sobre cobertura. Puede escogerse XML - facilita su posterior procesado - o texto plano - legible por humanos

Mapping scheme El tipo de aplanamiento a emplear. Consultar la documentación sobre la fase de análisis (3.3) para más información

Filter unused variables Filtrar los invariantes de variables que no han sido usadas

Integrate Simplify Ejecutar Simplify [Det05] sobre el conjunto de invariantes calculados, de forma que se simplifiquen las relaciones

Generate metrics Determina si mostrar o no las métricas de las declaraciones

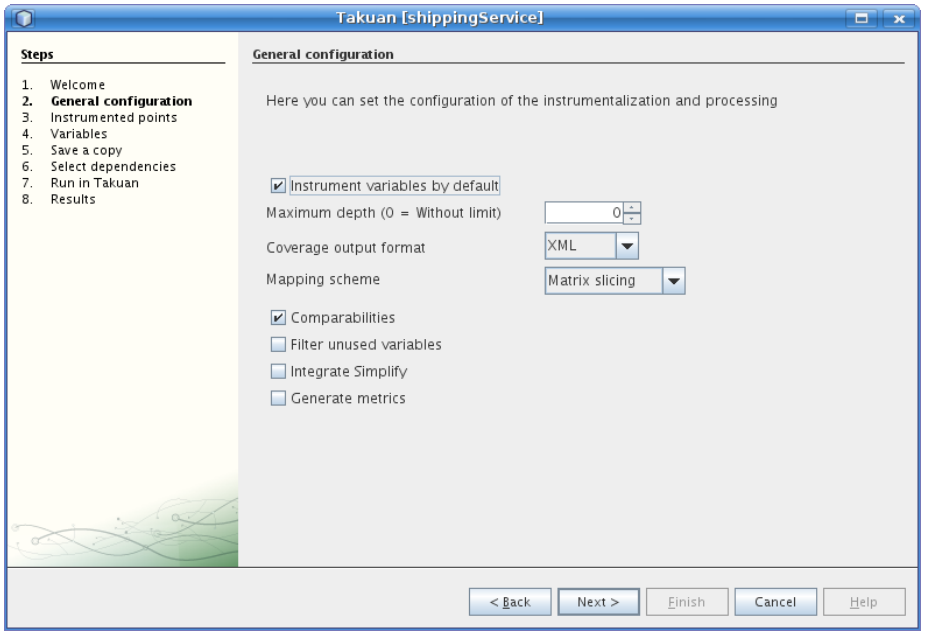

Figura 18.5: Parametrización

18.2.5. Variables y puntos de programa

En el siguiente diálogo se debe especificar cuál es la acción por defecto para todos los puntos del programa: instrumentar o no. Después, se especificará los puntos del programa que serán inspeccionados o no (lo opuesto al comportamiento general seleccionado).

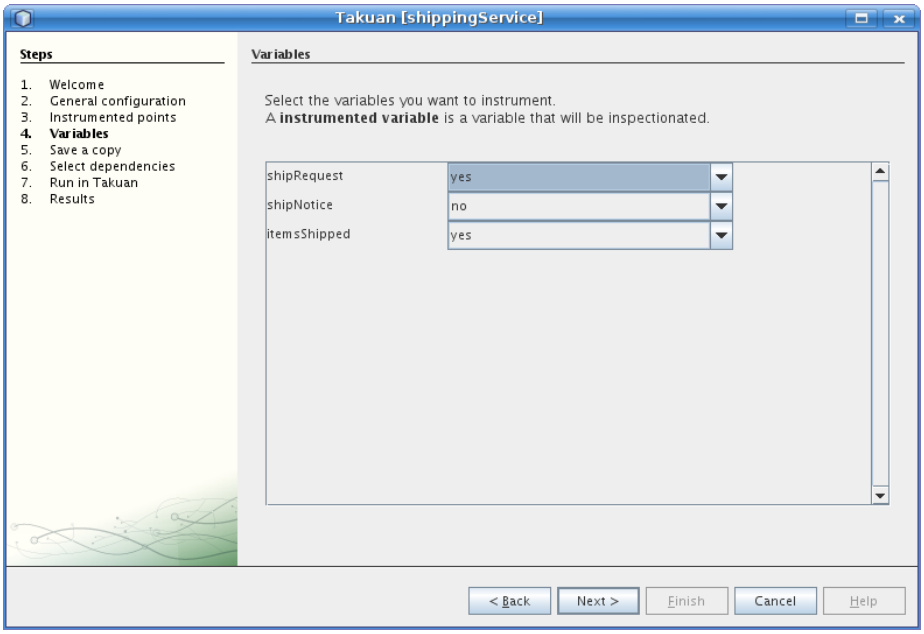

Figura 18.6: Puntos de programa

Una vez seleccionado los puntos de programa a inspeccionar, se hará lo mismo con las variables. Los valores admitidos son tres:

<not set> Se aplicará el comportamiento por defecto configurado en el primer paso del asistente

Yes Se instrumentará e inspeccionará la variable

No No se instrumentará ni inspeccionará la variable

18.2.6. Guardar una copia

Los cambios realizados sobre el fichero BPEL - parametros, puntos y variables a instrumentar e inspeccionar, ... - se perderán al realizar la ejecución. Si se desea, se puede guardar una copia del fichero BPEL modificado.

18.2.7. Dependencias

En el siguiente diálogo se especificarán los ficheros de los que depende la composición WS-BPEL, y el conjunto de casos de prueba.

Los botones + y - pueden emplearse para añadir o eliminar dependencias, incluyendo ficheros externos al proyecto.

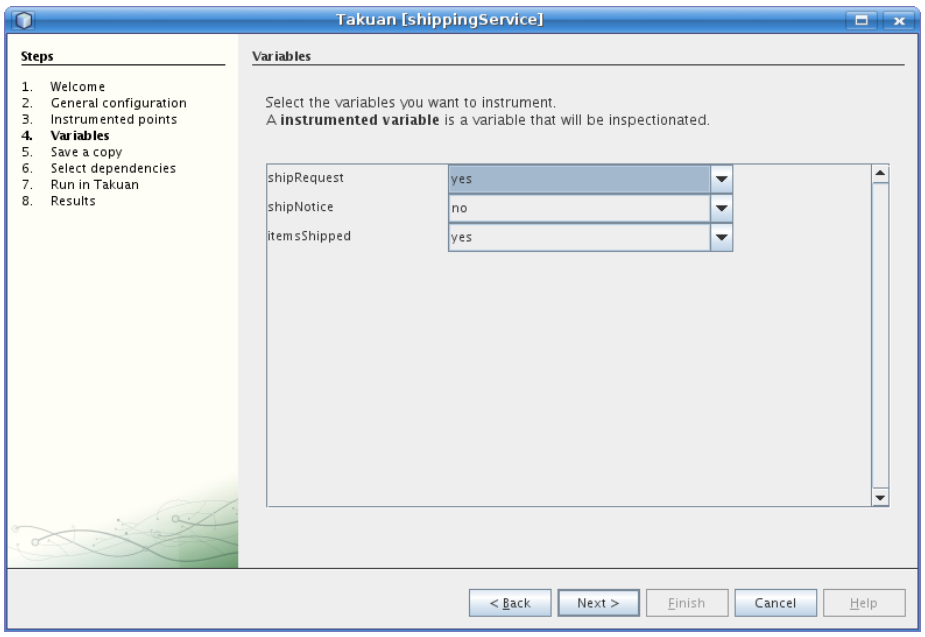

Figura 18.7: Detalle del diálogo de instrumentación de variables

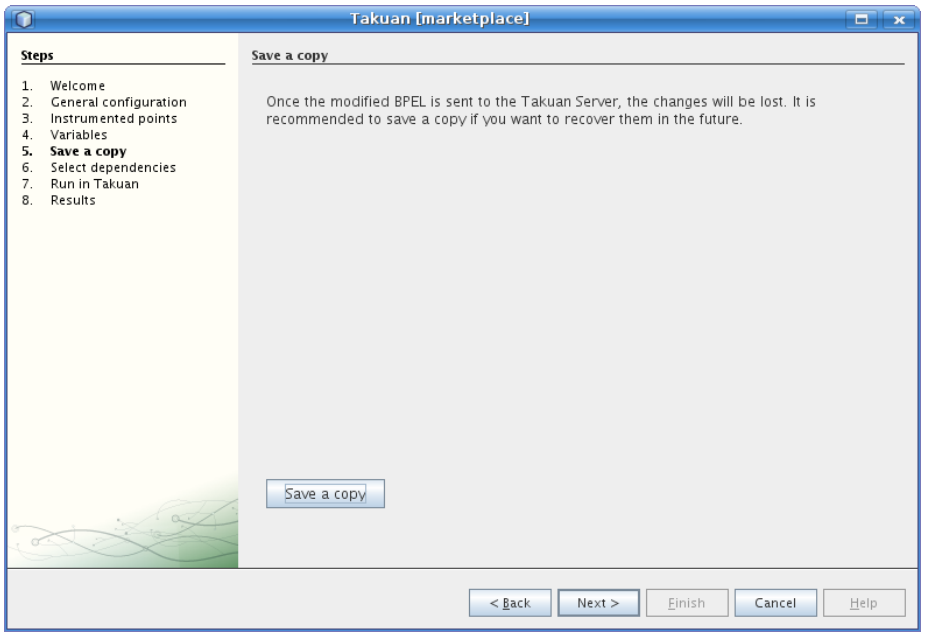

Figura 18.8: Guardar una copia

18.2.8. Ejecución

La última opción es la URL del servlet de Takuan, al que el asistente se conectará.

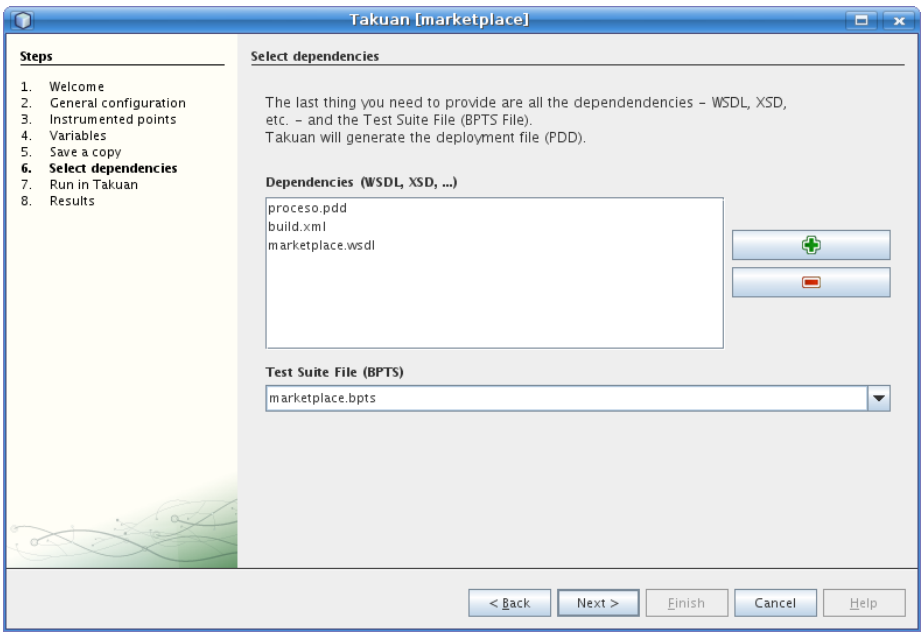

Figura 18.9: Diálogo de dependencias

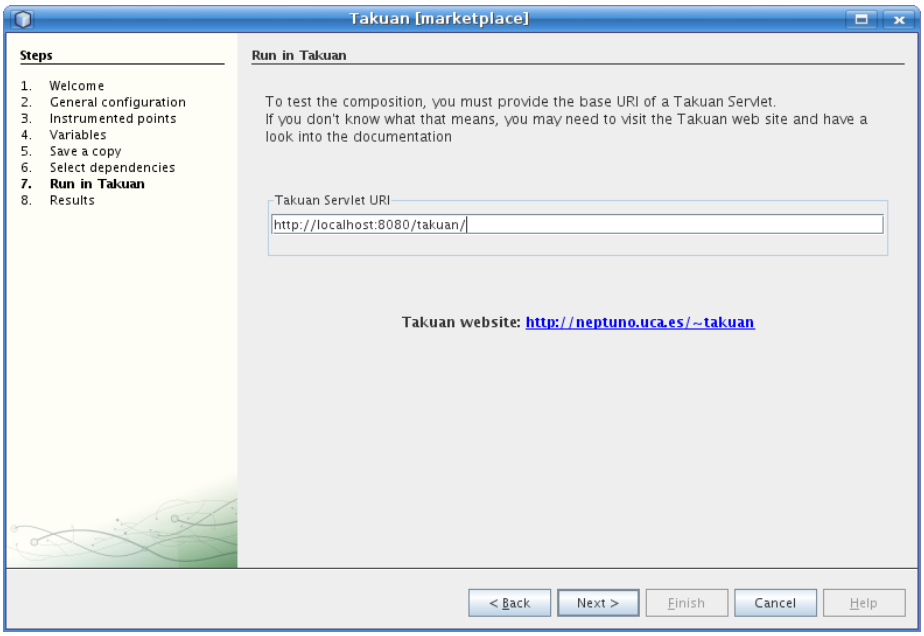

Figura 18.10: URL del servlet

Una vez introducido el valor, se mostrará el diálogo de progreso.

Figura 18.11: Diálogo de progreso

18.2.9. Resultados

Se obtienen dos tipos de resultados: invariantes e información sobre la cobertura. Las opciones disponibles son:

- Visualizarse
- Añadirse al proyecto BPEL
- Guardarse en un fichero externo

Nota: La opción de visualización está desactivada para la información de cobertura al ser un archivo comprimido con varios ficheros. Sin embargo:

- Cuando se añada al proyecto, el archivo se descomprimirá y se añadirá el contenido
- Cuando se seleccione la opción de guardar, se guardará el fichero ZIP descargado

Para saber cómo interpretar los resultados, puede consultar la sección 17.2.

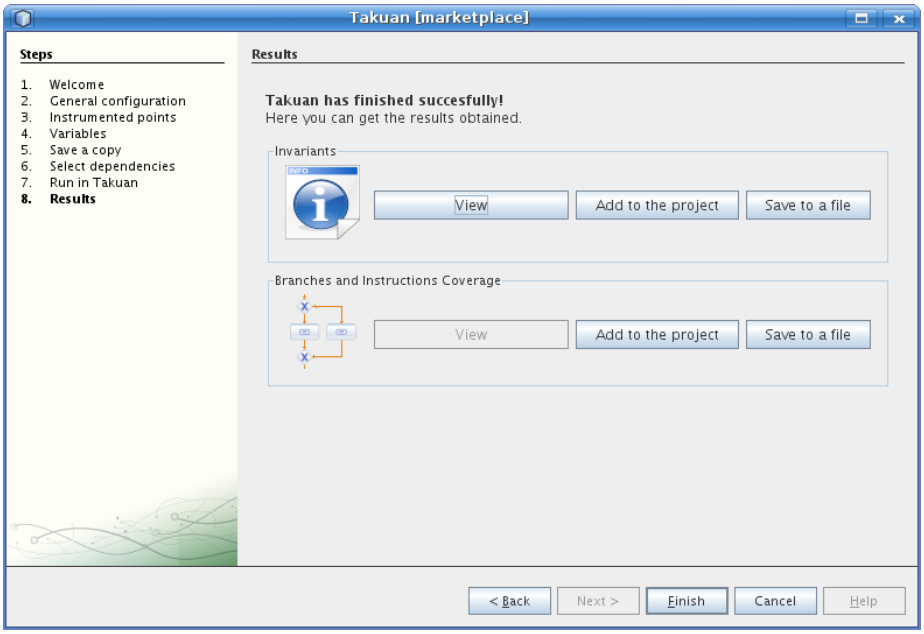

Figura 18.12: Resultados

Parte IV

Apéndices

Apéndice A

Guía rápida de WSDL

Este apéndice es un extracto de *Web Services Description Language (WSDL) Version 1.1* [W3C01]

A.1. Introducción

WSDL (*Web Services Description Language*) es un lenguaje que proporciona un modelo y un formato XML para describir servicios web, separando la descripción de la funcionalidad ofrecida por un servicio en concreto, de cómo y dónde se ofrece.

En este apéndice se hablará sobre la versión 1.1 de WSDL, ya que es la que se emplea en los ejemplos tomados como base para las pruebas de Takuan. Sin embargo, actualmente la versión recomendada por la W3C es la 2.0 [W3C07b], que es incompatible con la 1.1.

La notación seguida en la especificación de los elementos y atributos es como sigue:

***** El elemento o atributo anterior puede aparecer ninguna o varias veces

+ El elemento o atributo anterior puede aparecer varias veces, pero una como mínimo

? El elemento o atributo anterior puede aparecer una o ninguna vez

... Se omiten ciertos atributos no obligatorios

A.1.1. Descripción de un servicio

En un documento de este tipo se describen los servicios en dos niveles: uno abstracto y otro concreto.

En el nivel abstracto se describe un servicio a través de los mensajes que envía y recibe. Estos mensajes se describen empleando un lenguaje de definición de tipos, normalmente XML Schema.

Una *operación* asocia un patrón de intercambio de mensajes con uno o más mensajes. Es decir, especifica el número de mensajes enviados y/o recibidos, al igual de a quién se le envía o de quién se recibe.

Una *interfaz* agrupa operaciones sin relacionarlas con el formato que se empleará en la comunicación.

En el nivel concreto, un enlace (*binding*) define los detalles del formato en el que se establecerá la conexión para una o más interfaces.

Un *"endpoint"* asocia una dirección de red con un "binding".

Finalmente, un *servicio* agrupa "endpoints" que implementan una interfaz común.

En definitiva, una descripción de servicios indica como los potenciales clientes deben interactuar con el servicio que es descrito. Por otra parte, una interfaz describe una interacción potencial (pero no requerida), y una operación define cómo se supone que la interacción tendrá lugar en caso de que ocurra, pero no obliga a realizarla.

A.2. Definición del servicio

A.2.1. Estructura del documento

Un documento WSDL es un conjunto de definiciones.

Los servicios se definen usando seis elementos principales:

Tipos Proporcionan las definiciones de tipos de datos usados para describir los mensajes.

Mensajes Representan una definición abstracta de los datos transmitidos.

Tipos de puertos Es un conjunto de operaciones abstractas, donde cada operación refiere un mensaje de entrada y otro de salida.

Vínculos Especifica el protocolo específico y el formato de los datos empleados en las operaciones y mensajes de un tipo de puerto en concreto.

Puertos Especifica una dirección para un vínculo.

Servicios Empleado para crear un conjunto de puertos relacionados.

Los documentos WSDL pueden tener asignados un atributo opcional con un nombre identificativo. Opcionalmente, también puede asignarse un atributo *targetNamespace* de tipo URI cuyo valor NO debe ser relativo.

Modularidad WSDL permite asociar un espacio de nombres con un ficheros usando una sentencia <import>:

```
<definitions .... >
    <import namespace="uri" location="uri"/> *
</definitions>
```

El uso de elementos de importación permite separar los diferentes elementos de la definición de un servicio en documentos independientes que podrán ser importados cuando sea necesario. Esta técnica mejora la claridad del código, además de facilitar la reutilización de definiciones.

Extensiones WSDL permite ser extendido con nuevos elementos asociados a una tecnología específica, cuyo espacio de nombres debe ser diferente al de WSDL. Esto permite innovar sin necesidad de revisar la base de las especificaciones de WSDL [W3C01].

Documentación Opcionalmente puede emplearse el elemento <wsdl:document> como un contenedor para documentación en formato legible por humanos. Ese elemento puede estar dentro de cualquier elemento WSDL.

A.2.2. Tipos

Los elementos de tipos encierran definiciones de tipos de datos relevantes para los mensajes. Por cuestiones de interoperabilidad y neutralidad, WSDL emplea el uso de XSD (XML Schema) [W3C09] como el sistema de referencia, sin perjuicio de otros.

```
1 <definitions .... >
2     <types>
3         <xsd:schema .... />*
4     </types>
5 </definitions>
```

Este sistema de tipos puede usarse independientemente de si el formato final de intercambio es XML o no. Es conveniente seguir las siguientes recomendaciones:

- Usar elementos en lugar de atributos.
- No incluir atributos o elementos que son propios de la codificación de la comunicación (por ejemplo *soap:root*).
- Los tipos vectoriales deberían extender el tipo Array definido en SOAP v1.1. Usar el nombre *ArrayOfXXX* para los tipos de vectores, donde XXX es el tipo de los elementos.
- Usar el tipo *xsd:anyType* para representar un campo o parámetro que pueda tener cualquier valor.

A.2.3. Mensajes

Los mensajes están compuestos de una o más partes lógicas, cada una de ellas asociadas a un tipo definido previamente. En el caso de XSD, WSDL define varios atributos de tipado como:

element Para elementos XSD

type Para elementos simples o complejos XSD

La sintaxis para definir es un mensaje es:

```
1 <definitions .... >
2     <message name="nmtoken"> *
```

```
3           <part name="nmtoken" element="qname"? type="qname
                "?/> *
4       </message>
5   </definitions>
```

El atributo "name" del mensaje proporciona un nombre único entre todos los mensajes definidos dentro del documento WSDL.

El atributo "name" de la parte (<part>) proporciona un nombre único entre todas las partes del mensaje que las contiene.

Partes de un mensaje

Las "partes" son un mecanismo que permite describir el contenido lógico de un mensaje. A través del enlazado puede asociarse información dependiente del vínculo a una parte de un mensaje. Por ejemplo, si un mensaje se va a emplear con RPC, una parte podría representar un parámetro.

Se emplean múltiples partes si el mensaje tiene múltiples unidades lógicas.

Ejemplo de un mensaje con dos partes:

```
1  <definitions .... >
2      <types>...</types>
3      <message name="PO">
4          <part name="po" element="tns:PO"/>
5          <part name="invoice" element="tns:Invoice"/>
6      </message>
7  </definitions>
```

Mensajes concretos vs abstractos

Generalmente las definiciones de los mensajes se consideran abstractas. Será un vínculo (*binding*) el que describa como se mapea el contenido abstracto en un formato específico. Sin embargo, en algunos casos la definición abstracta se ajustará casi completamente, o exactamente, a uno o más vínculos, por los que esos vínculos proporcionarán muy poca o ninguna información de mapeo. Sin embargo, otro vínculo del mismo mensaje puede requerir mucha información de mapeo. Por ese motivo, hasta que no se examina el *binding* no puede determinarse si un mensaje es abstracto o no.

A.2.4. Tipos de puertos

Un tipo de puerto es un conjunto de operaciones abstractas y los mensajes relacionados.

```
1  <wsdl:definitions .... >
2      <wsdl:portType name="nmtoken">
3          <wsdl:operation name="nmtoken" .... /> *
4      </wsdl:portType>
5  </wsdl:definitions>
```

El atributo "name" proporciona un nombre único dentro de todos los tipos de puertos definidos dentro del documento WSDL.

El nombre de una operación se determina por el atributo "name".
WSDL tiene cuatro tipos de operaciones:

Único sentido El extremo recibe un mensaje.

Petición-respuesta El extremo recibe un mensaje y responde con otro correlacionado.

Solicitud-respuesta El extremo envía un mensaje y recibe otro correlacionado.

Notificación El extremo envía un mensaje.

A pesar de que la estructura básica de WSDL permite enlaces para estas cuatro primitivas, sólo define enlaces para "Único sentido" y "Petición-respuesta". Se espera que las especificaciones que definan los protocolos para "Solicitud-respuesta" o "Notificación" incluyan también extensiones de enlazado[1].

Operación de un único sentido

```
1 <wsdl:definitions .... > <wsdl:portType .... > *
2         <wsdl:operation name="nmtoken">
3             <wsdl:input name="nmtoken"? message="qname"/>
4         </wsdl:operation>
5     </wsdl:portType >
6 </wsdl:definitions>
```

El elemento <input> especifica el formato del mensaje abstracto para esta operación.

Operación de petición-respuesta

```
1 <wsdl:definitions .... >
2     <wsdl:portType .... > *
3         <wsdl:operation name="nmtoken" parameterOrder="
              nmtokens">
4             <wsdl:input name="nmtoken"? message="qname"/>
5             <wsdl:output name="nmtoken"? message="qname"/>
6             <wsdl:fault name="nmtoken" message="qname"/>*
7         </wsdl:operation>
8     </wsdl:portType >
9 </wsdl:definitions>
```

Los elementos <input> y <output> especifican los formatos de los mensajes para la petición y la respuesta respectivamente.

El elemento opcional <fault> especifica el formato del mensaje de error en caso de haberlo.

[1] Como no es relevante para este proyecto, se omitirán en adelante estos dos tipos de operaciones.

Nombres de los elementos dentro de una operación

El atributo "name" de los elementos <input> y <output> proporciona un nombre único dentro del tipo de puerto contenedor.

Para evitar tener que nombrar todos los elementos de entrada y salida, WSDL proporciona valores por defecto en función del nombre de la operación si el atributo "name" no se especifica, tomará el valor:

Único sentido El nombre de la operación.

Petición-respuesta El nombre de la operación con el valor "Request" y "Response" respectivamente.

Cada elemento <fault> debe ser nombrado para permitir a un vínculo el especificar el formato concreto del mensaje de error.

A.2.5. Enlazado

Un enlace (*binding*) define el formato y el protocolo de los mensajes y operaciones definidos por un <portType>. Puede haber cualquier número de enlaces para un mismo tipo de puerto.

```
1  <wsdl:definitions .... >
2      <wsdl:binding name="nmtoken" type="qname"> *
3          <-- Elemento de extension (1) --> *
4          <wsdl:operation name="nmtoken"> *
5              <-- Elemento de extension (2) --> *
6              <wsdl:input name="nmtoken"? > ?
7              <-- Elemento de extension (3) -->
8              </wsdl:input>
9              <wsdl:output name="nmtoken"? > ?
10             <-- Elemento de extension (4) --> *
11             </wsdl:output>
12             <wsdl:fault name="nmtoken"> *
13                 <-- Elemento de extension (5) --> *
14             </wsdl:fault>
15         </wsdl:operation>
16     </wsdl:binding>
17 </wsdl:definitions>
```

name Un nombre único

type El tipo de puerto

Los elementos de extensión se usan para especificar la gramática concreta para la entrada (3), salida (4) y mensajes de error (5). Además, también pueden especificarse enlaces por operación (2) y por enlace (1).

Dado que los nombres de las operaciones no tienen por qué ser únicos (por ejemplo, en el caso de sobrecarga), el atributo "name" de la operación puede no ser suficiente para identificar unívocamente la operación. En ese caso, se emplearán los atributo "name" de los elementos de entrada (<wsd:input>) y salida (<wsdl:output>) correspondientes.

Un enlace debe especificar un único protocolo, pero **no** información sobre direccionamiento.

A.2.6. Puertos

Un puerto define un extremo individual especificando una una dirección para un enlace.

```
1 <wsdl:definitions .... >
2     <wsdl:service .... > *
3         <wsdl:port name="nmtoken" binding="qname"> *
4             <-- Elemento de extension (1) -->
5         </wsdl:port>
6     </wsdl:service>
7 </wsdl:definitions>
```

name Un nombre único para el puerto.

binding Enlace (*binding*) asociado.

El elemento de extensión (1) se emplea para especificar la dirección del puerto.

Un puerto **no** debe tener más de una dirección. Tampoco debe suministrar más información que la dirección.

A.2.7. Servicios

Un servicio agrupa un conjunto de puertos.

```
1 <wsdl:definitions .... >
2     <wsdl:service name="nmtoken"> *
3         <wsdl:port .... />*
4     </wsdl:service>
5 </wsdl:definitions>
```

El atributo "name" debe ser único.

Los puertos contenidos en un servicio tienen las siguientes relaciones:

- Ninguno de los puertos se comunica con uno de los otros (la salida de uno de ellos no es la entrada de otro)
- Si un servicio tiene varios puertos que comparten tipo, pero emplean diferentes enlaces o direcciones, los puertos son alternativas. Es decir, proporcionan un comportamiento semánticamente similar, pero en base a protocolos diferentes.
- Permiten al usuario determinar si desea comunicarse con un servicio en base a si soporta o no varios tipos de puertos, en caso de que exista alguna relación implícita entre sus operaciones.

A.3. Tipos de enlazado

En este apéndice no se cubrirán los detalles de cada tipo de enlazado, aunque se describirán brevemente dos proporcionados por WSDL: SOAP y HTTP.

A.3.1. Enlazado con SOAP

WSDL incluye un enlace con SOAP 1.1, soportando las siguientes características:

- Indicar que un enlace está asociado al protocolo SOAP 1.1
- Especificar la dirección de un extremo SOAP
- El URI para la cabecera HTTP del mensaje SOAP
- Especificar una lista de definiciones para cabeceras que son enviadas como parte del envoltorio SOAP

En el ejemplo incluido en la sección A.4 puede verse un caso de uso de SOAP.

A.3.2. Enlazado con HTTP

WSDL incluye soporte para enlazar con HTTP 1.1, usando los métodos GET y POST para describir las interacciones entre el navegador y el sitio web. Esto permite a otras aplicaciones distintas al navegador el interactuar con el sitio. Se puede especificar:

- Una indicación de si se debe usar GET o POST
- Una dirección para el puerto
- Una dirección relativa para cada operación (relativa a la dirección base definida por el puerto)

A.4. Ejemplo

Este ejemplo está extraido del estándar WSDL 1.0 [W3C01].

```
1  <?xml version="1.0"?>
2  <definitions name="StockQuote"
3    targetNamespace="http://example.com/stockquote.wsdl"
4    xmlns:tns="http://example.com/stockquote.wsdl"
5    xmlns:xsd1="http://example.com/stockquote.xsd"
6    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7    xmlns="http://schemas.xmlsoap.org/wsdl/">
8
9    <types>
10     <schema targetNamespace="http://example.com/
           stockquote.xsd"
11           xmlns="http://www.w3.org/2000/10/XMLSchema">
12       <element name="TradePriceRequest">
```

```
13          <complexType>
14            <all>
15              <element name="tickerSymbol" type="string"/>
16            </all>
17          </complexType>
18        </element>
19        <element name="TradePrice">
20          <complexType>
21            <all>
22              <element name="price" type="float"/>
23            </all>
24          </complexType>
25        </element>
26      </schema>
27    </types>
28
29    <message name="GetLastTradePriceInput">
30      <part name="body" element="xsd1:TradePriceRequest"/>
31    </message>
32
33    <message name="GetLastTradePriceOutput">
34      <part name="body" element="xsd1:TradePrice"/>
35    </message>
36
37    <portType name="StockQuotePortType">
38      <operation name="GetLastTradePrice">
39        <input message="tns:GetLastTradePriceInput"/>
40        <output message="tns:GetLastTradePriceOutput"/>
41      </operation>
42    </portType>
43
44    <binding name="StockQuoteSoapBinding" type="
          tns:StockQuotePortType">
45      <soap:binding style="document"
46                    transport="http://schemas.xmlsoap.org/
                         soap/http"/>
47        <operation name="GetLastTradePrice">
48          <soap:operation soapAction="http://example.com/
              GetLastTradePrice"/>
49            <input>
50              <soap:body use="literal"/>
51            </input>
52            <output>
53              <soap:body use="literal"/>
54            </output>
55        </operation>
56    </binding>
57
58    <service name="StockQuoteService">
59      <documentation>My first service</documentation>
```

```
60        <port name="StockQuotePort" binding="
              tns:StockQuoteBinding">
61          <soap:address location="http://example.com/
                stockquote"/>
62        </port>
63    </service>
64
65 </definitions>
```

Apéndice B

Guía rápida de WS-BPEL

Este apéndice es un extracto de *Web Services Business Process Execution Language* [OAS07].

La notación seguida en la especificación de los elementos y atributos es como sigue:

***** El elemento o atributo anterior puede aparecer ninguna o varias veces

+ El elemento o atributo anterior puede aparecer varias veces, pero una como mínimo

? El elemento o atributo anterior puede aparecer una o ninguna vez

... Se omiten ciertos atributos no obligatorios

B.1. Introducción

El objetivo de los servicios web es lograr la interoperabilidad entre aplicaciones empleando estándares web. Emplean un modelo de integración débilmente acoplado para permitir una integración flexible de sistemas heterogéneos en una variedad de dominios que incluyen negocio-consumidor, negocio-negocio y la integración de aplicaciones empresariales. Los estándares iniciales que definían los servicios web fueron: SOAP [W3C07a], WSDL [W3C01] (Ver apéndice A), y UDDI [OAS05].

Sin embargo, la integración de sistemas requiere algo más que la capacidad de realizar interacciones simples a través de protocolos estándar. El pleno potencial de los servicios web como plataforma de integración será alcanzada sólo cuando las aplicaciones y los procesos de negocios puedan integrar sus complejas interacciones usando un modelo estándar de integración de procesos. El modelo suministrado por WSDL sólo permite modelos sin estado de tipo petición-respuesta, o de un sólo sentido.

Los modelos de interacción de negocios normalmente suponen secuencias de mensajes entre pares, tanto de tipo petición-respuesta, como de sentido único, contenidos en interacciones con estado que involucran a dos o más partes. Para definir dicho tipo de interacciones, se necesita un método formal para describir este intercambio de mensajes. Un *proceso abstracto* puede ser usado para describir el comportamiento visible de de un proceso de negocio en sus interacciones,

pero sin revelar la implementación interna. Hay dos motivos para separar estos aspectos públicos de los internos o privados:

1. No se quiere revelar los detalles internos de la toma de decisiones y manejo de datos a la otra parte.

2. Aún no siendo este el caso, se consigue libertad para modificar los aspectos privados de la implementación del proceso, sin interferir con su comportamiento público.

El comportamiento observable debe ser descrito de forma independiente de la plataforma, y mostrar aspectos que pueden tener relevancia en interacciones entre negocios.

Se deben tener en cuenta los siguientes aspectos para describir un proceso de negocio:

- Los procesos de negocios muestran un comportamiento dependiente de los datos. Por ejemplo, un proceso de suministro depende de datos como el número de elementos de un pedido, el valor total, etcétera.

- La habilidad para especificar condiciones excepcionales y sus consecuencias, incluyendo secuencias de recuperación, es tan importante como la posibilidad de definir el comportamiento en caso de éxito.

- Las interacciones de mucho tiempo de vida incluyen muchas unidades de trabajo (a veces anidadas), cada una con sus requisitos respecto a los datos.

También se debe poder distinguir entre dos tipos de procesos: abstractos y ejecutables.

Abstracto Es un proceso parcialmente especificado, que no está destinado a ser ejecutado, y que debe ser declarado explícitamente como abstracto. Un proceso abstracto puede ser que oculte detalles concretos de las operaciones realizadas.

Esta ocultación puede hacerse:

- Explícitamente, con *tokens* opacos
- Por omisión

Ejecutable Es un proceso que está completamente definido, y, por tanto, puede ser ejecutado.

Como se ha mencionado antes, es posible usar WS-BPEL para definir un proceso de negocio ejecutable. Los mecanismos para ello se basan exclusivamente en Servicios Web y datos en formato XML. Estos procesos se ejecutan e interactúan con sus pares de forma consistente, independientemente de la plataforma sobre la que se ejecute.

En definitiva, WS-BPEL define un modelo y una gramática para describir el comportamiento de un proceso de negocio basándose en las interacciones entre el proceso y sus pares. Estas interacciones se realizan a través de interfaces de

Servicios Web. El proceso WS-BPEL define cómo se coordinan las interacciones entre múltiples servicios para lograr un objetivo, así como la lógica y los estados necesarios para lograr esta coordinación. También incorpora mecanismos para tratar con excepciones y fallos de procesamiento. Es más, WS-BPEL incluye un mecanismo para definir cómo una actividad individual o compuesta será compensada en casos de error.

WS-BPEL emplea especificaciones XML como: WSDL 1.1 [W3C01], XML Schema 1.0 [W3C09], XPath 1.0 [W3C07c], y XSLT 1.0 [W3C99].

B.2. Estructuras

A continuación se describirán brevemente y de forma no exhaustiva las estructuras que proporciona WS-BPEL para la composición web.

B.2.1. Estructura básica

Como ya se ha comentado, WS-BPEL es un lenguaje basado en XML. Su espacio de nombres es

```
1 http://docs.oasis-open.org/wsbpel/2.0/process/executable
```

El elemento raíz es <process>, y debe definirse, como mínimo, el atributo `name`: el nombre del proceso. Adicionalmente, pueden especificarse los atributos:

queryLanguage Especifica el lenguaje de consulta que usas el proceso para la selección de nodos en las asignaciones. El valor por defecto es `"urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"`, es decir, XPath 1.0 [W3C07c].

expressionLanguage Determina el lenguaje de las expresiones usadas en el elemento <process>. El valor por defecto es `"urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"`, que, de nuevo, representa XPath 1.0.

suppressJoinFailure Especifica si el error "joinFailure" será suprimido para todas las actividades. El valor por defecto es "no". Puede sobrecargarse por cada actividad.

exitOnStandardFault Si el valor de este atributo es "yes", entonces el proceso deberá terminar inmediatamente (como si se alcanzara una actividad <exit>), cuando ocurra un fallo estándar distinto de bpel:joinFailure. Si el valor es "no", entonces el proceso podrá manejar un fallo estándar usando un manejador de fallos. El valor por defecto es "no".

En el caso de que se estén empleando extensiones, podrá añadirse un elemento <extensions> al proceso dentro del cual se especifiquen tantas extensions (<extension>) como sea necesario. Estos elementos deben tener dos atributos:

namespace El espacio de nombres de la extensión

mustUnderstand Especifica si el motor de ejecución debe entender la extensión, o puede ignorarla[1]

Además de las extensiones, un proceso puede contener los siguientes elementos en su primer nivel:

import Importación de otros ficheros: declaraciones WSDL, XSD, etcétera

partnerLinks Declaración de los enlaces a otros servicios

variables Declaración de variables globales

correlationSets Vincula un mensaje con una instancia concreta del proceso. Se emplea, por ejemplo, en mensajes asíncronos

faultHandlers Manejadores de fallos

eventHandlers Manejadores de eventos

Una actividad La actividad principal del proceso. Puede ser un flujo, una secuencia,...

La mayoría de estos elementos se detallan a continuación. Para obtener una información más completa, puede consultar [OAS07].

B.2.2. Enlace con otros ficheros

Para enlazar un proceso WS-BPEL con otros documentos - definiciones XSD, ficheros WSDL, ... - se emplea el elemento <import>, añadido justo dentro del elemento raíz.

La forma de uso habitual sigue el esquema:

```
1 <import namespace="espacioDeNombres"
2     location="uriDelFichero"?
3     importType="tipoDeImport"/>
```

namespace El espacio de nombres que se está importando - targetNamespace del fichero importado.

location Ruta o nombre del fichero (p.e. "ship.xsd")

importType El tipo del fichero importado. Se debe especificar el espacio de nombres del tipo. Por ejemplo:

XML Schema http://www.w3.org/2001/XMLSchema

WSDL http://schemas.xmlsoap.org/wsdl/

[1] Takuan añade una extensión al proceso inspeccionado para especificar que las extensiones de Takuan deben ser ignoradas.

B.2.3. Partner Links

Los servicios con los que interactúa un proceso se modelan como *partner links* en WS-BPEL. Cada partnerLink se caracteriza por un partnerLinkType (ver apéndice A para más detalles). Por supuesto, puede haber más de un partnerLink con el mismo tipo.

```
<partnerLinks>
    <partnerLink name="NCName"
        partnerLinkType="QName"
        myRole="NCName"
        partnerRole="NCName"
        initializePartnerRole="yes|no" />
</partnerLinks>
```

name El nombre del partnerLink. Debe ser único.

partnerLinkType El tipo del partnerLink. Estará definido en un WSDL.

myRole El rol que realiza la composición WS-BPEL.

partnerRole El rol que realiza el agente externo.

initializePartnerRole Si su valor es "yes", el motor WS-BPEL inicializará el partnerRole antes de que se use. En caso contrario, se inicializará cuando se use por primera vez. Puede omitirse, en cuyo caso el comportamiento no está definido.

Un partnerLink puede definirse dentro de un elemento <process> o dentro de un elemento <scope>.

B.2.4. Variables

Las variables permiten almacenar mensajes que formen parte del estado de un proceso de negocio. Hay tres tipos de variables:

- Tipo de mensaje WSDL
- Tipo XML Schema, simple o complejo
- Elemento XML Schema

La sintaxis para la declaración de variables es:

```
<variables>
    <variable name="BPELVariableName"
        messageType="QName"?
        type="QName"?
        element="QName"?>+
        from-spec
    </variable>
</variables>
```

Dentro del elemento <variables> pueden declararse tantas variables como sea necesario.

name El nombre de la variable. Debe de ser único dentro de su ámbito

messageType Se usará este atributo si el tipo de la variable es un tipo de mensaje WSDL

type Se usará este atributo si el tipo de la variable es un tipo XML Schema

element Se usará este atributo si el tipo de la variable es un elemento XML Schema

from-spec Valor con el que debe inicializarse la variable. Es opcional.

B.2.5. Correlaciones

Puede pensarse que el destino de los mensajes enviados a un proceso de negocio es un puerto definido en el fichero WSDL. Sin embargo, esto es una ilusión, ya que en realidad los servicios con estado son instanciados para actuar de acuerdo al historial de interacciones. Por tanto, estos mensajes no sólo son enviados al puerto correcto, sino también a una instancia concreta del proceso de negocio. Los mensajes que crean una nueva instancia son un caso especial.

En el paradigma orientado a objetos, estas interacciones con estado son manejadas por referencias a objetos, lo que permite alcanzar una instancia concreta con el historial y estado adecuados. Sin embargo, en los servicios web el uso de estos tipos de referencias crearían una serie de dependencias que no sobrevivirían la evolución independiente de los detalles de implementación de cada servicio. Por tanto, se confía en los datos y las cabeceras del protocolo para evitar el uso de tokens dependientes de la implementación.

Por ejemplo, consideremos un caso en el que un comprador envía una orden de compra a un vendedor en la máquina destino. Este vendedor necesita devolver un mensaje de aceptación del pedido, y este mensaje debe enrutarse a la instancia correcta del vendedor. La forma estándar de hacer esto es usar un token en el mensaje de compra - como el ID de la orden - que es copiado al mensaje de aceptación para poder correlacionar. Este token puede estar en la envoltura del mensaje, en una cabecera o en el mismo documento. En cualquier caso, la exacta localización es fija e independiente de la instancia. Sólo el valor depende de la misma.

La declaración de las correlaciones se basan en propiedades de los mensajes. Una propiedad es simplemente un campo dentro de un mensaje identificado por una consulta, lo que sólo puede hacerse cuando la estructura está bien definida - usando XML Schema, por ejemplo.

Un conjunto de tokens de correlación se define como un conjunto de propiedades compartidas por todos los mensajes de un mismo grupo. Este conjunto de propiedades se llama “conjunto de correlación”.

```
1 <correlationSets>?
2     <correlationSet name="NCName" properties="QName-list"
          />+
3 </correlationSets>
```

Los atributos son

name El nombre del conjunto. Debe ser único.

properties Las propiedades compartidas. Deben ser definidas usando tipos simples XML Schema.

Las correlaciones pueden usarse en en cualquier actividad relacionada con mensajes (<receive>, <reply>, <onMessage>, <onEvent>, y <invoke>).

Una vez que se inicializa un conjunto de correlación, los valores de las propiedades de ese conjunto deben ser idénticos para todos los mensajes en todas las operaciones que usen ese conjunto de correlación.

```
1 <correlations>
2    <correlation  set="NCName"
3        initiate="yes|join|no"?
4        pattern="request|response|request-response"?  />+
5 </correlations>
```

set El nombre del conjunto de correlación definido previamente con correlationSet.

initiate Sus valores pueden ser

- **yes** La actividad relacionada debe intentar iniciar el conjunto de correlación. Si ya lo estuviera, se lanzará el fallo bpel:correlationViolation.
- **no** La actividad no debe intentar iniciar el conjunto. Si no lo había sido previamente, se lanzará la excepción bpel:correlationViolation. Si lo fue pero falla el requisito de consistencia, se lanzará la misma excepción.
- **join** La actividad relacionada debe intentar iniciar el conjunto de correlación si no lo ha sido anteriormente. Si ya había sido iniciado y falla el requisito de consistencia, se lanzará la excepción bpel:correlationViolation.

pattern Cuando se emplea uno o más conjuntos de correlación en una actividad <invoke> de tipo petición/respuesta, el atributo "pattern" se usa para indicar si la correlación se aplica al mensaje saliente (request), entrante (response) o ambos (request-response).

Si se especifica más de un conjunto de correlación, los requisitos de inicialización y consistencia se tendrán en cuenta para todos.

Para poder procesar los valores del conjunto de correlación, el motor WS-BPEL debe poder encontrar una definición <vprop:propertyAlias> y aplicarla al mensaje. Para ver cómo definir propiedades, consulte el apéndice A sobre WSDL.

Si el valor retornado fuera distinto que un único elemento o una colección de caracteres de información, se lanzará una excepción bpel:selectionFailure.

A continuación se muestra un ejemplo de definición de un conjunto de correlación.

```
<correlationSets xmlns:cor="http://example.com/
    supplyCorrelation">

    <!-- Order numbers are particular to a customer,
         this set is carried in application data -->

    <correlationSet name="PurchaseOrder"
        properties="cor:customerID cor:orderNumber" />

    <!-- Invoice numbers are particular to a vendor,
        this set is carried in application data -->

    <correlationSet name="Invoice"
        properties="cor:vendorID cor:invoiceNumber" />
</correlationSets>
```

Y su uso dentro de una actividad de recepción.

```
<receive partnerLink="Buyer" portType="SP:PurchasingPT"
    operation="PurchaseRequest" variable="PO">
    <correlations>
        <correlation set="PurchaseOrder" initiate="yes" />
    </correlations>
</receive>
```

B.2.6. Expresiones

WS-BPEL permite el uso de diferente tipos de expresiones:

Expresiones booleanas Deben devolver valores booleanos (true / false). Se usan en los elementos transition, join, while, y las condiciones de los if.

Expresiones "deadline" Deben devolver valores válidos del tipo xsd:date y xsd:dateTime. Se usan en las expresiones <until> de los elementos <onAlarm> y <wait>.

Expresiones de duración Deben devolver un valor del tipo xsd:duration. Se emplean en las expresiones de los elementos <onAlarm> y <wait>, y la expresión <repeatEvery> del elemento <onAlarm>.

Expresiones de enteros sin signo Deben devolver valores del tipo xsd:unsignedInt. Se usan en los elementos <startCounterValue>, <finalCounterValue>, y <branches> dentro de los <forEach>.

Expresiones generales Dentro de los elementos <asign>.

Si la expresión devolviera un tipo incorrecto, se lanzaría una excepción bpel:invalidExpressionValue.

Pueden usarse las expresiones XPath:

- boolean(object)

- string(object)
- number(object)

Adicionalmente, pueden usarse dos extensiones XPath propias de WS-BPEL:

getVariableProperty(variable, property) Devuelve el valor de una propiedad de una variable. Ambos parámetros son del tipo string.

doXslTransform(string, node-set, (string, object)*) El primer parámetro especifica, como una cadena XPath, la URI de la hoja de estilos que se va a aplicar. El segundo parámetro es un conjunto de nodos sobre el que se aplicará la conversión. Por último, un número arbitrario, y opcional, de parejas de:

- Una cadena XPath que especifica el nombre cualificado de un parámetro XPath
- Un objeto XPath que proporciona el valor del parámetro cuyo nombre especifica el parámetro anterior

Todas las expresiones deben seguir el estándar XPath 1.0 [W3C07c].

B.2.7. Asignaciones

La actividad <assign> puede ser usada para copiar datos de una variable a otra, y también para construir e insertar nuevos datos usando expresiones. El uso de estas expresiones viene motivado por la necesidad de realizar calculos simples (como incrementos). Pueden operar sobre variables, propiedades, y literales.

Otra posibilidad de las asignaciones es copiar referencias desde y hacia partnerLinks. Incluso es posible incluir operaciones definidas como elementos de extensión pertenecientes a espacios de nombres diferentes al de WS-BPEL. Si dicha extensión no es reconocida por el procesador WS-BPEL, y no está sujeta a `mustUnderstand="yes"`, se ignorará.

La forma más común de uso de una asignación es:

```
1 <assign validate="yes|no"?>
2   <copy keepSrcElementName="yes|no"?
3         ignoreMissingFromData="yes|no"?>
4     from-spec
5     to-spec
6   </copy>
7 </assign>
```

También puede usarse, en lugar del elemento copy, el elemento <extensionAssignOperation>, pero no vamos a cubrirlo aquí.

validate Se comprueba si los valores asignados concuerdan con las definiciones de tipo XML de las variables. Si se establece a "yes", y falla la validación, se lanzará un error `bpel:invalidVariables`.

keepSrcElementName Especifica si el nombre de elemento de destino (seleccionado por to) reemplazará el nombre del elemento de origen.

ignoreMissingFromData Si se establece a "yes", se suprime el error bpel:selectionFailure, en caso de estar accediendo a una variable no inicializada.

La especificación <from> puede ser de la forma:

Asignación desde un literal
```
<from><literal>literal value</literal></from>
```

Asignación desde una propiedad
```
<from variable="BPELVariableName" property="QName" />
```

Asignación desde un partnerLink
```
<from partnerLink="NCName"
      endpointReference="myRole|partnerRole" />
```

Asignación desde una expresión
```
<from expressionLanguage="anyURI"?>expression</from>
```

El atributo `expressionLanguage` es opcional. Por defecto, será XPath 1.0.

Desde una consulta
```
<from variable="BPELVariableName" part="NCName"?>
   <query queryLanguage="anyURI"?>?
      queryContent
   </query>
</from>
```

La especificación `to` puede ser de la forma:

Hacia una variable
```
<to variable="BPELVariableName" part="NCName"?/>
```

El atributo `part` se puede usar en caso de que el tipo de la variable sea un tipo de mensaje WSDL. En caso contrario, no debe emplearse.

Hacia una propiedad
```
<to variable="BPELVariableName" property="QName" />
```

Hacia un partnerLink
```
<to partnerLink="NCName" />
```

Hacia una expresión
```
<to expressionLanguage="anyURI"?>expression</to>
```

De nuevo, por defecto el lenguaje de la expresión será XPath 1.0.

B.2.8. Recibir y responder

Un proceso de negocio suministra servicios a través de mensajes entrantes. Dichos mensajes se capturan a través de actividades <receive> y se responderán con la actividad <reply>.

Receive

La actividad <receive> especifica el partnerLink que contiene el myRole usado para recibir mensajes, el tipo de puerto (opcional) y la operación que espera que el cliente invoque. Adicionalmente se define una variable para recibir los datos del mensaje. También puede usarse, alternativamente, el elemento fromPart.

La sintaxis es:

```
<receive partnerLink="NCName"
    portType="QName"?
    operation="NCName"
    variable="BPELVariableName"?
    createInstance="yes|no"?
    messageExchange="NCName"?>

    <correlations>?
        <correlation set="NCName" initiate="yes|join|no"? />+
    </correlations>

    <fromParts>?
        <fromPart part="NCName" toVariable="
            BPELVariableName" />+
    </fromParts>
</receive>
```

La única forma de crear una nueva instancia de un proceso es con el atributo "createInstance" con valor "yes" en una actividad <receive> o una <pick>. El valor por defecto es "no". Puede haber más de una actividad que cree una nueva instancia de una composición.

Reply

La actividad <reply> se emplea para mandar respuesta a solicitudes previamente aceptadas a través de una actividad como <receive>. Pueden mandarse respuestas de un sólo sentido invocando la actividad correspondiente en el partnerLink.

```
<reply partnerLink="NCName"
    portType="QName"? operation="NCName"
    variable="BPELVariableName"?
    faultName="QName"?
    messageExchange="NCName"?>

    <correlations>?
```

```
8       <correlation set="NCName" initiate="yes|join|no"? />+
9     </correlations>
10
11    <toParts>?
12      <toPart part="NCName" fromVariable="BPELVariableName" />+
13    </toParts>
14
15  </reply>
```

En una respuesta normal, el atributo faultName no se usa y el atributo variable (o su equivalente toPart) indica el mensaje de respuesta. Cuando la respuesta indica un fallo, se usa el atributo faultName, y el atributo variable, o su equivalente, contendrán el correspondiente fallo. El nombre del fallo debe referirse a uno definido en la operación usada en la actividad reply, y la variable coincidir con el tipo asociado.

B.2.9. Invocar otros servicios

Invoke

La actividad <invoke> se usa para llamar servicios web ofrecidos por proveedores de servicios. El uso típico es invocar una operación en un servicio externo. Una actividad invoke puede contener otras actividades dentro de un manejador de compensación de de fallos.

Las operaciones pueden ser de petición-respuesta o de un sólo sentido, dependiendo de la definición WSDL.

```
1  <invoke partnerLink="NCName"
2      portType="QName"?
3      operation="NCName"
4      inputVariable="BPELVariableName"?
5      outputVariable="BPELVariableName"?>
6
7      <correlations>?
8        <correlation set="NCName" initiate="yes|join|no"?
9          pattern="request|response|request-response"? />+
10     </correlations>
11
12     <catch faultName="QName"?
13       faultVariable="BPELVariableName"?
14       faultMessageType="QName"?
15       faultElement="QName"?>*
16       activity
17     </catch>
18
19     <catchAll>?
20       activity
21     </catchAll>
22
```

```
23    <compensationHandler>?
24        activity
25    </compensationHandler>
26
27    <toParts>?
28        <toPart part="NCName" fromVariable="
              BPELVariableName" />+
29    </toParts>
30
31    <fromParts>?
32        <fromPart part="NCName" toVariable="
              BPELVariableName" />+
33    </fromParts>
34
35 </invoke>
```

Llamadas de un sólo sentido sólo necesitan la variable de entrada (inputVariable) o equivalente, dado que no se espera ninguna respuesta.

B.2.10. Estructuras de control

WS-BPEL proporciona, como cualquier otro lenguaje de programación, estructuras que permiten controlar el flujo de ejecución del proceso.

Procesamiento secuencial

Una actividad <sequence> contiene una o más actividades que son ejecutadas de forma secuencial, en el orden en el que aparecen dentro de la misma. Esta actividad finaliza cuando finalice la última actividad que contenga.

Ejemplo:

```
1 <sequence>
2    <receive>...</receive>
3    <assign>...</assign>
4    <reply>...</reply>
5 </sequence>
```

Condicionales

La actividad <if> proporciona control de ramas. Consiste en una lista ordenada de una o más ramas condicionales definidas por el <if> y por <elseif> opcionales, seguidos por un <else> igualmente opcional.

Ejemplo:

```
1 <if xmlns:inventory="http://supply-chain.org/inventory"
2     xmlns:FLT="http://example.com/faults">
3
4     <condition>
5     bpel:getVariableProperty('stockResult','
          inventory:level')
6       >
```

```
7     100
8     </condition>
9
10    <flow>
11       <!-- Lista de actividades -->
12    </flow>
13    <elseif>
14       <condition>
15       bpel:getVariableProperty('stockResult','
             inventory:level')
16        >=
17       0
18       </condition>
19       <throw faultName="FLT:OutOfStock"
20              variable="RestockEstimate" />
21    </elseif>
22    <else>
23       <throw faultName="FLT:ItemDiscontinued" />
24    </else>
25 </if>
```

Bucles - While

La actividad <while> permite la ejecución repetida de una actividad. Dicha actividad, contenida dentro del elemento while, se ejecutará mientras la condición booleana se evalúe a verdadero (true).

Ejemplo:

```
1 <while>
2    <condition>$orderDetails > 100</condition>
3    <scope><secuence>...</secuence></scope>
4 </while>
```

Bucles - RepeatUntil

La actividad <repeatUntil> funciona de forma similar a while, con la diferencia de que el cuerpo se ejecutará al menos una vez.

Ejemplo:

```
1 <repeatUntil>
2    <scope><secuence>...</secuence></scope>
3    <condition>$orderDetails > 100</condition>
4 </repeatUntil>
```

Bucles - ForEach

La actividad <forEach> ejecutará su contenido N+1 veces, donde N es el valor de <finalCounterValue> menos <startCounterValue>.

Estructura:

```
<forEach counterName="BPELVariableName" parallel="yes|no"
    standard-attributes>
    standard-elements

    <startCounterValue expressionLanguage="anyURI"?>
        unsigned-integer-expression
    </startCounterValue>

    <finalCounterValue expressionLanguage="anyURI"?>
        unsigned-integer-expression
    </finalCounterValue>

    <completionCondition>?
        <branches expressionLanguage="anyURI"?
            successfulBranchesOnly="yes|no"?>?
            unsigned-integer-expression
        </branches>
    </completionCondition>
    <scope ...>...</scope>
</forEach>
```

Una vez que comienza la actividad forEach, se evalúan las expresiones en startCounterValue y finalCounterValue, permaneciendo su valor constante durante toda la ejecución de la actividad. Ambas expresiones deben devolver un valor entero sin signo (xsd:unsignedInt).

En caso de que uno, o los dos, valores devueltos no cumplan la condición, se lanzará una excepción bpel:invalidExpressionValue. En caso de que el valor de startCounterValue sea mayor que el de finalCounterValue, no se ejecutará la actividad.

La actividad hija de una actividad forEach siempre debe ser una actividad <scope>. <forEach> crea una variable implícita con el valor del contador, además de paralelismo dinámico - un número paralelo de ramas cuyo número no se puede saber a priori - .

Para más información sobre esta estructura, puede consultar la documentación del estándar WS-BPEL [OAS07].

Scope

Un elemento <scope> (ámbito) proporciona el contexto que influencia el comportamiento de la ejecución de las actividades contenidas en su interior. Esto incluye variables, partnerLinks, compensaciones, etc.

Cada <scope> tiene una actividad principal que define su comportamiento normal. Esta actividad puede ser una compleja (secuencia, condicional, bucle,...) con varios niveles anidados.

La sintaxis es:

```
<scope isolated="yes|no"? exitOnStandardFault="yes|no"?
    standard-attributes>
    standard-elements

```

```
5     <variables>?
6     </variables>
7
8     <partnerLinks>?
9     </partnerLinks>
10
11    <messageExchanges>?
12    </messageExchanges>
13
14    <correlationSets>?
15    </correlationSets>
16
17    <eventHandlers>?
18    </eventHandlers>
19
20    <faultHandlers>?
21    </faultHandlers>
22
23    <compensationHandler>?
24    </compensationHandler>
25
26    <terminationHandler>?
27    </terminationHandler>
28
29     activity
30
31  </scope>
```

Todos los manejadores contenidos en <scope> pueden acceder a todas las variables, partnerLinks, mensajes, y correlationSets definidos en el nuevo ámbito, y en todos sus antecesores.

Las variables, partnerLinks, mensajes y correlationSets definidos dentro de un ámbito son sólo accesibles dentro del mismo.

Procesamiento en paralelo

La actividad <flow> proporciona ejecución concurrente y sincronización. Una actividad de este tipo finaliza cuando todas sus actividades han terminado.

En el siguiente ejemplo hay dos actividades de envío que se inician de forma concurrente. Suponiendo que sean ambas llamas son de petición-respuesta, la actividad flow finalizará cuando se reciban respuestas de ambos servicios.

```
1  <sequence>
2
3     <flow>
4        <invoke partnerLink="Seller" ... />
5        <invoke partnerLink="Shipper" ... />
6     </flow>
7
8     <invoke partnerLink="Bank" name="transferMoney" ... />
9  </sequence>
```

Esta actividad también permite sincronizaciones entre las actividades contenidas en su interior, independientemente de la profundidad de anidamiento. El elemento <link> es el empleado para expresar estas dependencias. Sin embargo, su uso se escapa del objetivo meramente introductorio de este documento. Para más información, consulte el estándar [OAS07].

B.2.11. Excepciones

Manejo de errores

A veces una transacción puede fallar o ser cancelada, por problemas en la conexión o por cualquier otra razón. El trabajo parcial que ya se haya realizado debe deshacerse lo mejor posible, revirtiendo los efectos de actividades previas. WS-BPEL proporciona un sistema de compensación, de forma que se pueda controlar de forma flexible la reversión. Esto se consigue a través de manejadores de fallo y de compensación.

Manejadores de compensación

La posibilidad de declarar lógica de compensación además de la lógica habitual es la base del control de errores de WS-BPEL. Pueden usarse los ámbitos (<scope>) para delimitar la parte del código que debe ser reversible, definiendo un manejador de compensación.

Los manejadores de fallos y compensación pueden anidarse sin límite de profundidad.

Sintácticamente es simplemente una envoltura para una actividad que realiza la compensación.

```
1 <compensationHandler>
2     activity
3 </compensationHandler>
```

Manejadores de fallos

El manejo de fallos en un proceso de negocio puede entenderse como un cambio de modo dentro de un ámbito. El diseño del manejo de errores en WS-BPEL está diseñado para ser tratado como un "trabajo inverso", dado que su objetivo es deshacer el trabajo parcial, y fallido, realizado hasta la ocurrencia del error. La terminación de la actividad de un manejador de fallos, incluso cuando no relanza la excepción, no se considera una terminación con éxito del ámbito correspondiente.

La compensación no está activa en un ámbito si este tiene un manejador de fallos asociado.

Los manejadores de fallos explícitos, si se usan, asociados a un ámbito proporcionan un modo de definir un conjunto de actividades de manejo de fallos definidos por el usuario, definidos por elementos <catch> y <catchAll>. Cada elemento catch se define para atrapar un tipo específico de excepción. Puede proporcionar una variable opcional para contener datos asociados con el fallo. Si se omite el nombre del fallo, entonces el manejador atrapará todos los fallos con

el tipo de dato asociado. La variable en la que se almacenará el valor relacionado con el fallo se define con el atributo faultVariable. Esta variable está declarada implícitamente, y no es visible ni usable fuera del ámbito del manejador de fallo.

Adicionalmente, puede emplearse un elemento <catchAll> para capturar cualquier otra excepción que carezca de manejador propio.

Hay varias fuentes de error en WS-BPEL. Una respuesta fallida a una actividad invoke es una de ellas, donde el nombre del error y los datos asociados se basan en la definición del error definido en la operación WSDL. Una actividad throw es otra fuente, donde explícitamente se proporciona el nombre y los datos. Además, WS-BPEL define varios fallos estándares, y también pueden existir otros errores dependientes de la plataforma.

Un nombre de fallo puede usarse en un proceso WS-BPEL sin haber sido definido antes, por ejemplo, en una operación WSDL, o también puede omitirse.

Sintaxis:

```
<faultHandlers>
   <catch faultName="QName"?
      faultVariable="BPELVariableName"?
      ( faultMessageType="QName" | faultElement="QName" )
         ? >*
      activity
   </catch>
   <catchAll>?
      activity
   </catchAll>
</faultHandlers>
```

Debe existir al menos un elemento <catch> o <catchAll> dentro de un elemento <faultHandlers>.

Fallos predefinidos, compensación y manejadores de terminación

La visibilidad de los manejadores de compensación está limitada al ámbito (scope) que directamente lo contiene. Es por ello que la capacidad de compensar un ámbito se perdería si el ámbito no define ningún manejador. También hay que tener en cuenta que muchos errores no son respuestas programadas, ni el resultado de una invocación, por lo que no es razonable esperar un manejador explícito para todos los fallos en todos los ámbitos. Es por eso que WS-BPEL proporciona manejadores predeterminados cuando no se especifica ninguno. Lo mismo ocurre con los manejadores de compensación y con los de terminación.

El manejador por defecto de errores es:

```
<catchAll>
   <sequence>
      <compensate />
      <rethrow />
   </sequence>
</catchAll>
```

El manejador de compensación por defecto es:

```
<compensationHandler>
```

```
2     <compensate />
3 </compensationHandler>
```

Y el manejador de terminación por defecto es:

```
1 <terminationHandler>
2     <compensate />
3 </terminationHandler>
```

Manejar fallos estándares WS-BPEL

Si el valor del atributo exitOnStandardFault de un ámbito está definido a "yes", siempre que se lance un error WS-BPEL estándar distinto de bpel:joinFailure dentro del mismo ámbito el proceso *debe* terminar inmediatamente (como si se encontrara una actividad exit). Si el valor se establece a "no", entonces el proceso podrá manejar el fallo usando un manejador. El valor por defecto de este atributo es "no".

Cuando no se define, toma el valor del ámbito contenedor, o del elemento process.

Para más información sobre el comportamiento de estos manejadores, puede consultar el estándar WS-BPEL [OAS07].

Manejadores de eventos

Además de las excepciones producidas por errores, dentro de cada ámbito también existen eventos con sus correspondientes manejadores. Estos manejadores pueden ejecutarse de forma concurrente y son invocados cuando el evento asociado tiene lugar. La actividad hija de un manejador de evento *debe* ser una actividad de ámbito (scope).

Hay dos tipos de eventos:

- Mensajes entrantes correspondientes a operaciones WSDL
- Alarmas

La gramática para el conjunto de manejadores de eventos asociados a un ámbito es:

```
1  <eventHandlers>?
2      <onEvent partnerLink="NCName"
3          portType="QName"?
4          operation="NCName"
5          ( messageType="QName" | element="QName" )?
6          variable="BPELVariableName"?
7          messageExchange="NCName"?>*
8          <correlations>?
9              <correlation set="NCName" initiate="yes|join|no"
                   ? />+
10         </correlations>
11         <fromParts>?
12             <fromPart part="NCName" toVariable="
                   BPELVariableName" />+
```

```
13          </fromParts>
14          <scope ...>...</scope>
15      </onEvent>
16
17      <onAlarm>*
18
19          (
20          <for expressionLanguage="anyURI"?>duration-expr</
                for>
21          |
22          <until expressionLanguage="anyURI"?>deadline-expr</
                until>
23          )?
24
25          <repeatEvery expressionLanguage="anyURI"?>?
26              duration-expr
27          </repeatEvery>
28          <scope ...>...</scope>
29      </onAlarm>
30  </eventHandlers>
```

Un manejador de eventos debe tener al menos un elemento <onEvent>, o uno <onAlarm>.

Los manejadores de eventos se consideran parte del comportamiento normal de un ámbito, al contrario que los manejadores de error, compensación o terminación.

La actividad contenida por el elemento onEvent u onAlarm debe ser <scope>.

B.2.12. Otras

Wait

La actividad <wait> especifica un retraso de un determinado periodo de tiempo, o hasta que cierta fecha límite se alcance. En el caso de que la duración sea 0 o negativa, o si la fecha de <until> es anterior a la actual, la actividad finalizará inmediatamente.

Sintaxis:

```
1  <wait standard-attributes>
2      <for expressionLanguage="anyURI"?>
3        expresion-de-duracion
4      </for>
5      |
6      <until expressionLanguage="anyURI"?>
7        expresion-de-limite
8      </until>
9  </wait>
```

U ejemplo típico es invocar una operación en un momento concreto:

```
1  <sequence>
2      <wait>
```

```
3       <until>'2002-12-24T18:00+01:00'</until>
4    </wait>
5    <invoke partnerLink="CallServer"
6       portType="AutomaticPhoneCall"
7       operation="TextToSpeech"
8       inputVariable="seasonalGreeting" />
9 </sequence>
```

Empty

En ocasiones es necesario incluir una actividad que no haga nada. Por ejemplo, en el caso de que queramos capturar una excepción e ignorarla. También puede emplearse como punto de sincronización en un <flow>.

```
1 <empty standard-attributes/>
```

Exit

La actividad <exit> se emplea para finalizar inmediatamente la instancia del proceso. Todas las actividades en ejecución se terminan en el acto, sin lanzar ningún manejador de terminación, fallo o compensación.

```
1 <exit standard-attributes/>
```

Pick

La actividad <pick> espera a que tenga lugar un evento de un conjunto de ellos. Cuando lo recibe ejecutala actividad asociada con ese evento, e ignora el resto hasta que termine el <pick>.

Ejemplo:

```
1  <pick>
2
3    <onMessage partnerLink="buyer"
4       portType="orderEntry"
5       operation="inputLineItem"
6       variable="lineItem">
7       <!-- activity to add line item to order -->
8    </onMessage>
9
10   <onMessage partnerLink="buyer"
11      portType="orderEntry"
12      operation="orderComplete"
13      variable="completionDetail">
14      <!-- activity to perform order completion -->
15
16   </onMessage>
17   <!-- set an alarm to go off
18         3 days and 10 hours after the last order line -->
19
```

```
20      <onAlarm>
21          <for>'P3DT10H'</for>
22          <!-- handle timeout for order completion -->
23      </onAlarm>
24  </pick>
```

B.3. Ejemplo

El siguiente ejemplo es MarketPlace, extraído de ActiveVOS [End].

B.3.1. Código

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <process
3   xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
        executable"
4   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/
        process/executable"
5   xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/
        business-process/"
6   xmlns:tns="http://docs.active-endpoints.com/activebpel/
        sample/wsdl/marketplace/2006/09/marketplace.wsdl"
7   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8   name="marketplace"
9   targetNamespace="http://docs.active-endpoints.com/
        activebpel/sample/wsdl/marketplace/2006/09/
        marketplace.wsdl">
10
11   <import importType="http://schemas.xmlsoap.org/wsdl/"
12    location="marketplace.wsdl"
13    namespace="http://docs.active-endpoints.com/activebpel/
          sample/wsdl/marketplace/2006/09/marketplace.wsdl" />
14     <partnerLinks>
15         <partnerLink myRole="sales" name="seller"
16            partnerLinkType="tns:salesplnk" />
17         <partnerLink myRole="buying" name="buyer"
18                      partnerLinkType="tns:buyingplnk" />
19     </partnerLinks>
20     <variables>
21         <variable messageType="tns:sellerInfoMessage"
22                   name="sellerInfo" />
23         <variable messageType="tns:negotiationMessage"
24                   name="negotiationOutcome" />
25         <variable messageType="tns:buyerInfoMessage"
26                   name="buyerInfo" />
27     </variables>
28     <correlationSets>
29      <correlationSet name="negotiationIdentifier"
```

```
30                    properties="tns:negotiatedItem"/>
31    </correlationSets>
32    <sequence name="MarketplaceSequence">
33       <flow name="MarketplaceFlow">
34          <receive createInstance="yes"
35                   name="SellerReceive"
36                   operation="submit"
37                   partnerLink="seller"
38                   portType="tns:sellerPT"
39                   variable="sellerInfo">
40             <correlations>
41                <correlation set="negotiationIdentifier"
42                             initiate="join"/>
43             </correlations>
44          </receive>
45          <!-- tets -->
46          <receive createInstance="yes"
47                   name="BuyerReceive"
48                   operation="submit"
49                   partnerLink="buyer"
50                   portType="tns:buyerPT"
51                   variable="buyerInfo">
52             <correlations>
53                <correlation initiate="join"
54                             set="negotiationIdentifier"/>
55             </correlations>
56          </receive>
57       </flow>
58       <if name="MarketplaceSwitch">
59          <condition>
60          ($sellerInfo.askingPrice &lt;= $buyerInfo.offer)
61          </condition>
62          <assign name="SuccessAssign">
63             <copy>
64                <from>'Deal Successful'</from>
65                <to part="outcome"
66                    variable="negotiationOutcome"/>
67             </copy>
68          </assign>
69          <else>
70             <assign name="FailedAssign">
71                <copy>
72                   <from>'Deal Failed'</from>
73                   <to part="outcome"
74                       variable="negotiationOutcome"/>
75                </copy>
76             </assign>
77          </else>
78       </if>
79       <reply name="SellerReply" operation="submit"
```

```
80                partnerLink="seller" portType="tns:sellerPT"
81                variable="negotiationOutcome"/>
82        <reply name="BuyerReply" operation="submit"
83                partnerLink="buyer" portType="tns:buyerPT"
84                variable="negotiationOutcome"/>
85    </sequence>
86 </process>
```

B.3.2. Vista gráfica

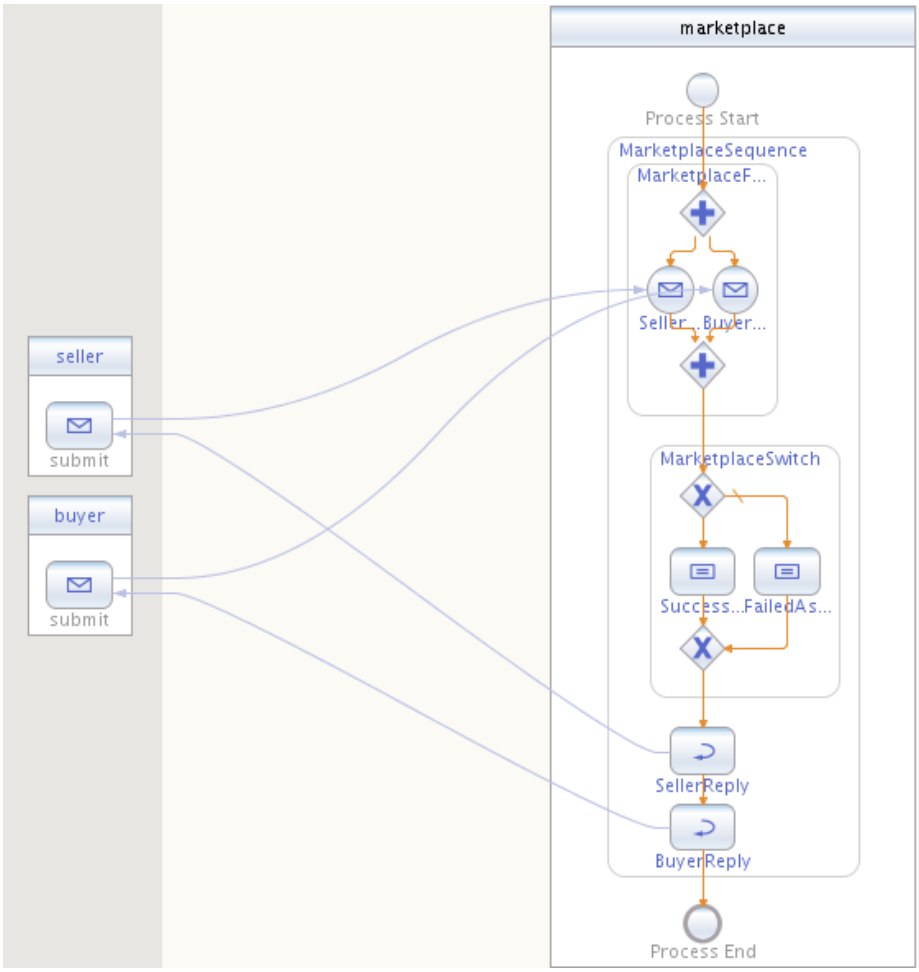

Figura B.1: Ejemplo MarketPlace

Apéndice C

IDEs

En este apéndice se describen brevemente algunos de los entornos integrados de desarrollo (IDE) disponibles para WS-BPEL.

C.1. DBE Studio

DBE Studio [Par] es un IDE para composiciones WS-BPEL implementado sobre la versión 3.1 de la plataforma Eclipse. Incluye una serie de editores, herramientas y asistentes para analizar y definir servicios web y composiciones WS-BPEL.

Fue desarrollado dentro del proyecto Digital Business Ecosystem, financiado por la Comisión Europea, siendo sus miembros colaboradores Waterford Institute of Technology, University of Central England, University of Surrey, Intel, Technical University of Crete, Trinity College Dublin y Soluta.net

Este entorno resultó inicialmente interesante porque decía tener un generador automático de casos de prueba. Sin embargo, actualmente el proyecto está abandonado, pues el último envío al repositorio tuvo lugar el 22 de junio de 2007, y no funciona con la última versión de Eclipse.

Es incompatible con el estándar WS-BPEL 2.0, que es sobre el que trabaja Takuan.

Por ambos motivos, se descartó su uso.

C.2. Active VOS

ActiveVOS [End] también está desarrollado sobre Eclipse. Es el IDE desarrollado por la misma compañía que mantiene el motor WS-BPEL que Takuan emplea, Active Endpoints, por lo que su compatibilidad es total.

Sin embargo, aunque la licencia del motor de ejecución es GPL, el editor es de código cerrado.

C.3. NetBeans

NetBeans [Mica, Micb] es una plataforma de desarrollo inicialmente concebida para proyectos Java, pero que gracias a diversos plugins, sirve de IDE para

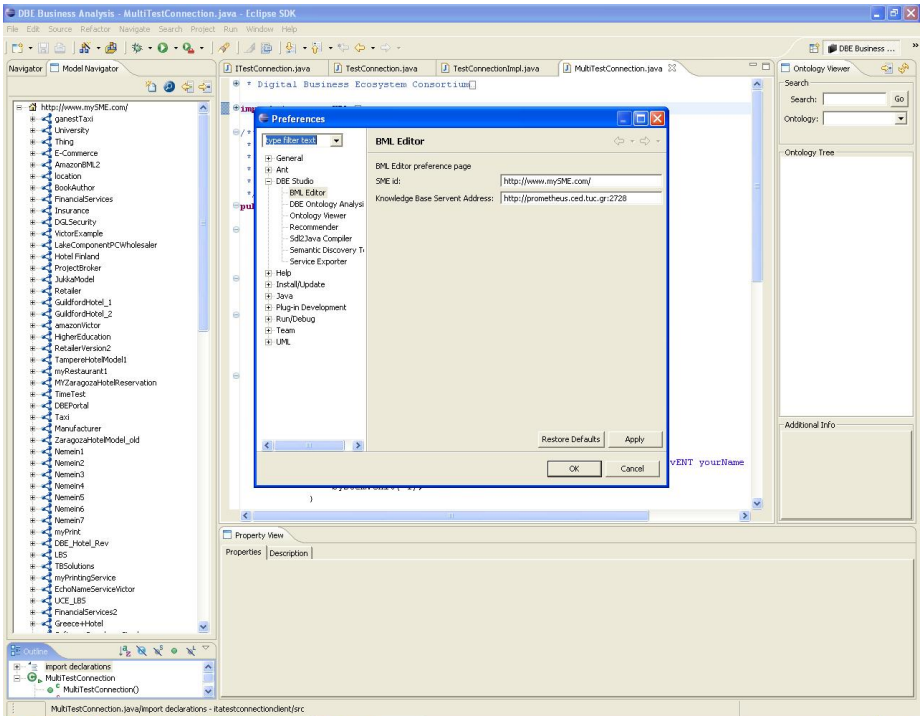

Figura C.1: Captura de DBE Studio

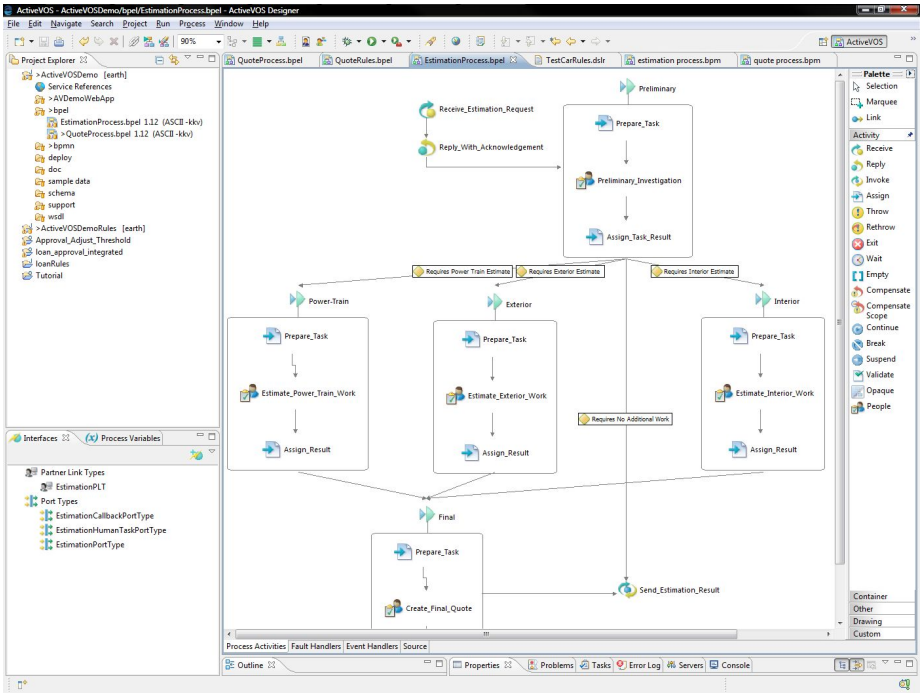

Figura C.2: Captura de ActiveVOS

multitud de tecnologías y lenguajes.

En el caso que nos ocupa, el editor para composiciones de servicios web y WS-BPEL (*Enterprise pack*) es uno de los más avanzados actualmente dentro del mundo del software libre. La facilidad para desarrollar nuevos plugins lo hace aún más atractivo, dado que permite integrar Takuan, o parte, en un IDE.

Puede consultar información sobre el plugin de Takuan para NetBeans 6.5.1 o superior en el capítulo 9.

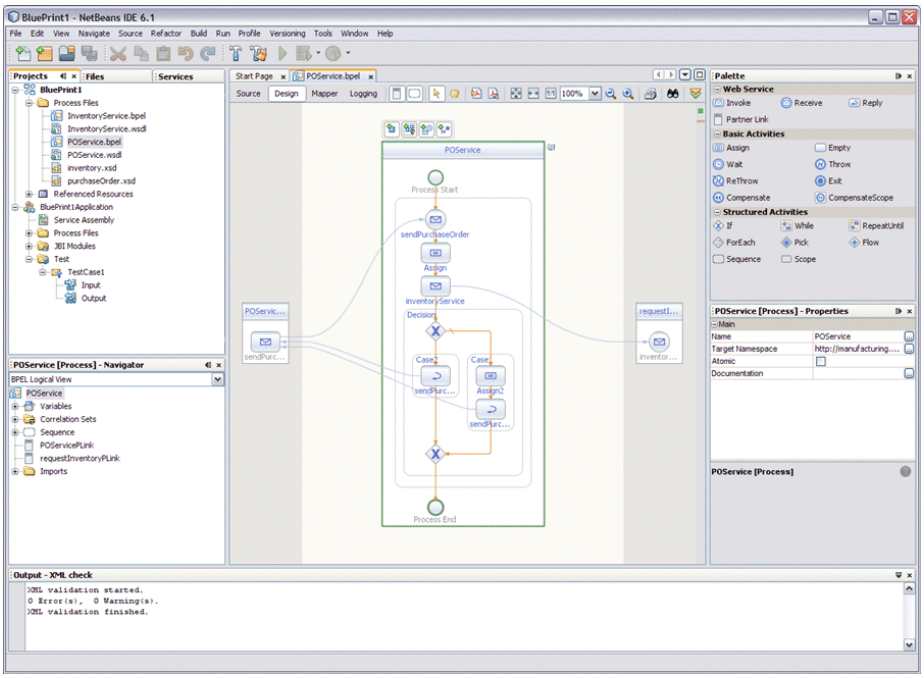

Figura C.3: Captura del editor BPEL de NetBeans

C.4. Eclipse

Eclipse [Ecl] es otro entorno de desarrollo con un sistema muy versátil de plugins distribuido bajo licencia libre. Su framework de desarrollo de añadidos es muy potente y versátil, aunque más difícil de usar inicialmente.

Es por este motivo, como se explica en el capítulo 9, se escogió NetBeans como plataforma para el plugin de Takuan.

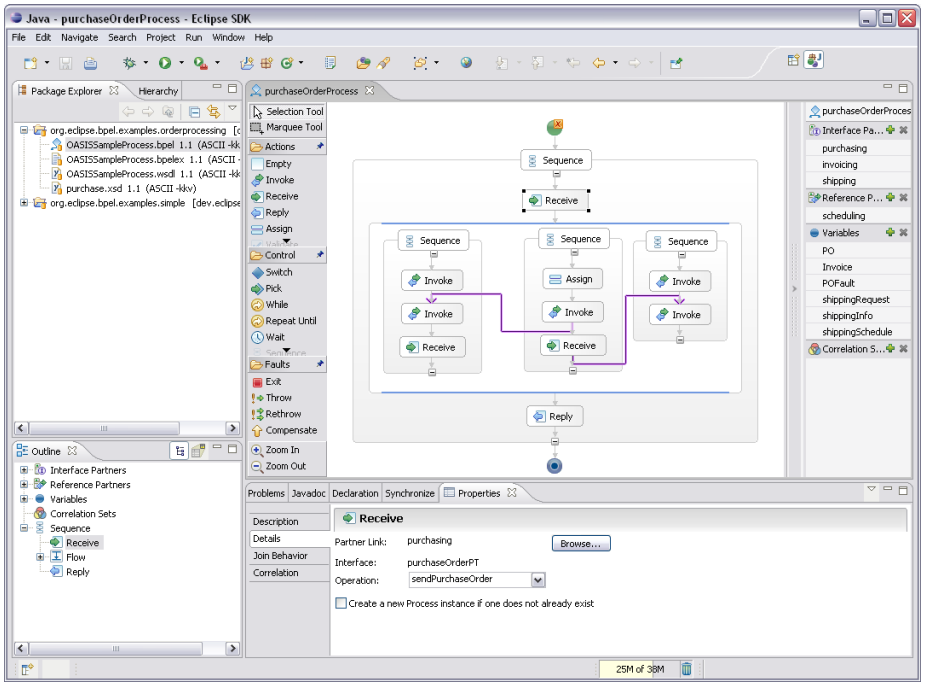

Figura C.4: Captura del editor BPEL de Eclipse

Apéndice D

Servidor Neptuno y máquina virtual de Takuan

D.1. Software instalado

- Distribución GNU/Linux Debian Etch, con kernel 2.6.18 x86_64
- Servidor web Apache 2.2.3
  - auth_basic
  - dav_fs
  - dav_svn
  - perl
  - php5
  - proxy_balancer
  - proxy
  - proxy_http
  - rewrite
- VirtualBox 2.0.2
- Redmine 0.7.3, funcionando sobre Rails 2.1.1 y Ruby 1.8.6 p114

D.2. Servidor web

D.2.1. Redmine

Redmine es un software libre de gestión de proyectos que usa el framework Ruby On Rails.

El servicio está siendo proporcionado por un clúster de tres servidores Mongrel - servidor web para aplicaciones escritas en Ruby - que se encuentran escuchando los puertos 8000, 8001 y 8002.

Los datos sobre los proyectos y usuarios se guardan en un servidor MySQL, mientras que a los repositorios Subversion accede a través del servidor Apache que hay funcionando.

Se ha desactivado el registro manual de usuarios, por lo que es necesario que un administrador de de alta a los nuevos.

Se encuentra instalado en `/srv/redmine`

La URL es `http://neptuno.uca.es/redmine`

¿Por qué Redmine?

Anteriormente se estaba empleando el sistema Trac para la gestión de tareas de investigación en Takuan: desarrollo de código, escritura colaborativa de artículos, web pública, etc. Pero la dificultad para administrarlo vía web - sobre todo el manejo de varios proyectos simultaneamente - impulsó la búsqueda de otras plataformas similares más flexibles y sencillas de administrar. Una de las opciones fue Redmine, y dado que un miembro del grupo ya lo conocía (Antonio García Domínguez) y lo recomendaba encarecidamente, fue elegido para ser instalado en el nuevo servidor.

Redmine permite gestionar los proyectos vía web, la creación automática de repositorios y usar los permisos de Redmine para el repositorio, cosa que Trac requiere que se haga de forma manual.

D.2.2. Subversion

Los repositorios se encuentran almacenados en `/srv/svn`

El acceso al exterior se proporciona a través del servidor Apache, usando la configuración de Redmine para el manejo de los permisos de lectura y escritura.

La URL base es `http://neptuno.uca.es/svn`

¿Por qué Subversion?

Subversion es un sistema de control de versiones muy maduro y extendido. Fue el que inicialmente se escogió para mantener el repositorio de los artículos, por lo que se ha seguido con él.

Ciertamente Git ha alcanzado una estabilidad más que fiable, pero no se ha estimado oportuno migrar los repositorios.

D.2.3. Apache

Como se ha mencionado, Apache funciona como servidor de SVN - a través de WebDAV - y como proxy reverso del cluster de tres instancias de Mongrel, estando también configurado como balanceador de carga.

¿Por qué Apache?

Por su madurez, implantación, flexibilidad, y capacidad de funcionar tanto como servidor para Subversion como balanceador de carga y servidor web normal, todo a la vez, de forma eficiente y sencilla de configurar.

¿Por qué Mongrel?

Existe un servidor web para aplicaciones Ruby llamado WEBrick que viene por defecto en toda instalación de Ruby, pero su eficiencia y estabilidad no es comparable a la de Mongrel, además de la posibilidad de crear clústers con Mongrel.

D.3. Máquina virtual

Takuan se encuentra instalado en una máquina virtual, con un sistema Ubuntu 8.04 para servidores. Dicha máquina virtual se encuentra configurada para que los cambios realizados se almacenen en un fichero temporal, y no sobre la imagen del disco, por lo que siempre se podrá disponer de un sistema limpio y recién instalado cuando se quiera.

Puede accederse a la máquina virtual de tres formas:

1. Servidor Apache Tomcat en el puerto 8080 - http://neptuno.uca.es:8080
2. Servidor SSH en el puerto 2222 - ssh takuan@neptuno.uca.es -p 2222
3. A través de RDP en el puerto 3389

Si quisiera iniciar la máquina de nuevo, partiendo de la instalación limpia, los pasos a seguir desde el sistema anfitrión son:

```
$ VboxManage controlvm TakuanVM poweroff
$ VBoxManage modifyvm TakuanVM -hda none
$ VBoxManage modifyvm TakuanVM -hda /srv/virtual/VDI/TakuanVM.vdi
```

El disco es de expansión dinámica, y está limitado a 20 GB. Sin embargo, el sistema virtual emplea LVM, por lo que se podría ampliar añadiendo otra imagen de disco.

Requiere VirtualBox ¿1.6.0

D.3.1. ¿Por qué VirtualBox?

Porque tiene una versión que es software libre, es gratuito, muy estable y considerablemente eficiente. Además, el fabricante distribuye drivers para mejorar el funcionamiento de los sistemas GNU/Linux virtualizados, compartir portapapeles, etc.

D.4. Otros servicios

También puede accederse al servidor a través de SSH por el puerto 22. Permite autenticación por pares de clave pública / privada.

D.5. Tareas programadas

Actualmente existen cuatro tareas programadas, ejecutándose todas ellas el domingo de madrugada:

1. Copia de seguridad de los repositorios
2. Copia de seguridad de la base de datos
3. Grabación de un DVD con las copias de seguridad
4. Reinicio de la máquina virtual

D.6. Problemas encontrados

D.6.1. Instalación de Redmine

Inicialmente se disponía de la versión empaquetada de Ruby para Debian, la 1.8.5. Sin embargo, esta versión no funciona bien con Rails 2.0.2, que es el framework sobre el que trabaja Redmine, por lo que hubo que desinstalar el paquete y buscar una versión compatible.

Fue necesario descargar el código de la versión 1.8.6, instalar las dependencias y compilar, aunque hubo varios problemas con el primer código descargado, principalmente con el soporte de MD5, por lo que hubo que volver a descargar y compilar una revisión posterior de la misma versión (la 114).

Una vez instalado Ruby, se procedió a la instalación de Gems – un gestor de paquetes para Ruby - y, por último, empleando Gems se instaló Rails 2.1.1.

D.6.2. Problemas no solucionados

No se ha conseguido hacer funcionar la comunicación con el SAI, ni por puerto serie ni USB.

Se han intentado tres opciones y ninguna ha funcionado:

- Instalar el software que proporciona el fabricante. El sistema es de 64 bits y el empaquetado de 32. Existen dependencias no resolubles.
- Compilar el software del fabricante (es GPL). Son necesarias determinadas versiones más recientes de algunas librerías, y actualizarlas lleva a actualizar casi el sistema completo por dependencias.
- Usar un software genérico alojado en los repositorios: upsd. No hemos conseguido que se comunicara. La documentación consultada parece sugerir que el problema surge por tener una versión del núcleo antigua.

Índice de figuras

Índice de cuadros

Bibliografía

[Act08] ActiveVOS. Activebpel WS-BPEL and BPEL4WS engine, febrero 2008. `http://sourceforge.net/projects/activebpel`.

[Apa] Apache Foundation. Apache Ant. `http://ant.apache.org/`.

[Bar07] Charlton Barreto. Web services business process execution language version 2.0 - primer. mayo 2007. `http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf`.

[Ber05] Antonia Bertolino y Eda Marchetti. A brief essay on software testing. En Richard H. Thayer y Mark Christensen, editores, *Software Engineering, The Development Process*. Wiley-IEEE Computer Society Press, third edición, 2005. ISBN 0471684171.

[Bis] Marc Bischof, Oliver Kopp y Tammo van Lessen. BPELscript. `http://www.bpelscript.org/`.

[Bjø97] Nikolaj Bjørner, Anca Browne y Zohar Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):páginas 49–87, 1997.

[Buc07] Antonio Bucchiarone, Hernánn Melgratti y Francesco Severoni. Testing service composition. En *Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE'07)*. 2007.

[Col03] Michael Colón, Sriram Sankaranarayanan y Henny Sipma. Linear invariant generation using non-linear constraint solving. En Warren A. Hunt Jr. y Fabio Somenzi, editores, *CAV*, tomo 2725 de *Lecture Notes in Computer Science*, páginas 420–432. Springer, 2003. ISBN 3-540-40524-0.

[Com07] OASIS Web Services Business Process Execution Language (WSBPEL) Technical Committee. Web services business process execution language version 2.0. abril 2007. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`.

[Det05] David Detlefs, Greg Nelson y James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):páginas 365–473, 2005.

[Dom07] Juan José Domínguez Jiménez, Antonia Estero Botaro, Inmaculada Medina Bulo, Manuel Palomo Duarte y Francisco Palomo Lozano. El reto de los servicios web para el software libre. En *Proceedings of the*

FLOSS International Conference 2007, páginas 117–132. Servicio de Publicaciones de la Universidad de Cádiz, Jerez de la Frontera, marzo 2007. ISBN 978-84-9828-124-8.

[Dom09] Antonio García Domínguez. Instalación de Takuan, 2009.

[Ecl] Eclipse Foundation. Eclipse.org.

[End] Active Endpoints. ActiveVOS. `http://www.activevos.com/`.

[Ern01] Michael D. Ernst, Jake Cockrell, William G. Griswold y David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):páginas 99–123, febrero 2001.

[Ern07] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz y Chen Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3):páginas 35–45, diciembre 2007.

[Gar08a] Antonio García Domínguez, Manuel Palomo Duarte y Inmaculada Medina Bulo. Framework para la generación dinámica de invariantes en composiciones de servicios web con WS-BPEL. En *Actas de Talleres de Ingeniería del Software y Bases de Datos*, páginas 1–10. SISTEDES, Gijón, España, octubre 2008.

[Gar08b] Antonio García Domínguez, Manuel Palomo Duarte y Inmaculada Medina Bulo. Implementación de un framework para la generación dinámica de invariantes en composiciones de servicios web con WS-BPEL. En *Actas de las IV Jornadas Científico-Técnicas en Servicios Web y SOA*, páginas 91–96. Sevilla, España, octubre 2008.

[Gar09] Antonio García Dominguez, Manuel Palomo Duarte, Inmaculada Medina Bulo y Alejandro Álvarez Ayllon. Takuan: generación dinámica de invariantes en composiciones de servicios web con WS-BPEL. septiembre 2009.

[Gas05] Cristophe Gaston y Dirk Seifert. Model-based testing of reactive systems. *Model-Based Testing of Reactive Systems*, páginas 293–322, 2005.

[Haa04] Hugo Haas y Allen Brown. Web services glossary. febrero 2004. `http://www.w3.org/TR/ws-gloss/`.

[IET] IETF. HTTP status pages. `http://tools.ietf.org/wg/http/`.

[May06a] Bob May y Dmitry Markovski. Developer guide to BPEL designer: The BPEL project, octubre 2006. `http://www.netbeans.org/kb/55/bpel_gsg_project.html`.

[May06b] Philip Mayer y Daniel Lübke. Towards a BPEL unit testing framework. En *TAV-WEB'06: Proceedings of the 2006 workshop on Testing, Analysis, and Verification of Web Services and Applications*, páginas 33–42. ACM, New York, NY, USA, 2006. ISBN 1-59593-458-8.

[Mica] Sun Microsystems. Netbeans. `http://www.netbeans.org/`.

[Micb] Sun Microsystems. Netbeans SOA application. `http://www.netbeans.org/kb/trails/soa.html`.

[Micc] Sun Microsystems. Virtualbox. `http://www.virtualbox.org/`.

[Mye04] Glenford J. Myers. *The Art of Software Testing*. John Wiley & Sons, second edición, 2004. ISBN 0471469122.

[OAS05] OASIS. UDDI standard 3.0. `http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`, febrero 2005.

[OAS06] OASIS. Reference model for service oriented architecture 1.0. febrero 2006.

[OAS07] OASIS. WS-BPEL 2.0 standard. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, abril 2007.

[Pal08] Manuel Palomo Duarte, Antonio García Domínguez y Inmaculada Medina Bulo. An architecture for dynamic invariant generation in WS-BPEL web service compositions. En *Proceedings of ICE-B 2008 - International Conference on e-Business*. INSTICC Press, Porto, Portugal, julio 2008.

[Pal09a] Manuel Palomo Duarte, Antonio García Domínguez y Inmaculada Medina Bulo. Enhancing WS-BPEL dynamic invariant generation using XML Schema and XPath Information. En *ICWE'09: Proceedings of the 9th International Conference on Web Engineering*, tomo 5648 de *Lecture Notes in Computer Science*, páginas 469–472. Springer, 2009. ISBN 978-3-642-02817-5.

[Pal09b] Manuel Palomo Duarte, Alejandro Álvarez Ayllón, Antonio García Domínguez y Inmaculada Medina Bulo. La cobertura de los casos de prueba en la generación dinámica de invariantes en composiciones WS-BPEL. *IV Taller sobre Pruebas en Ingeniería del Software*, páginas 1–8, septiembre 2009.

[Par] DBE Studio Partners. DBE studio. `http://dbestudio.sourceforge.net/`.

[PD08a] Manuel Palomo Duarte, Antonio García Domínguez y Inmaculada Medina Bulo. Improving Takuan to analyze a meta-search engine WS-BPEL composition. En *SOSE '08: Proceedings of the 2008 IEEE International Symposium on Service-Oriented System Engineering*, páginas 109–114. IEEE Computer Society, Washington, DC, USA, 2008. ISBN 978-0-7695-3499-2.

[PD08b] Manuel Palomo Duarte, Antonio García Domínguez y Inmaculada Medina Bulo. Takuan: A dynamic invariant generation system for WS-BPEL compositions. En *ECOWS '08: Proceedings of the 2008 Sixth European Conference on Web Services*, páginas 63–72. IEEE Computer Society, Washington, DC, USA, 2008. ISBN 978-0-7695-3399-5.

[Ram07] Isabel Ramos Román, Pablo Javier Tuya González y Javier Dolado Cosín. Técnicas cuantitativas para la gestión en la ingeniería del software. página 373, 2007.

[W3C99] W3C. XSL Transformations (XSLT). `http://www.w3.org/TR/xslt`, noviembre 1999.

[W3C01] W3C. WSDL 1.1. `http://www.w3.org/TR/wsdl`, marzo 2001.

[W3C07a] W3C. SOAP 1.2 recommendation. `http://www.w3.org/2000/xp/Group`, abril 2007.

[W3C07b] W3C. WSDL 2.0. `http://www.w3.org/TR/wsdl20/`, junio 2007.

[W3C07c] W3C. XSL Recommendations (including XPath 1.0 and 2.0). `http://www.w3.org/Style/XSL`, enero 2007.

[W3C09] W3C. W3C XML Schema Definition Language 1.1. `http://www.w3.org/TR/xmlschema11-1/`, abril 2009.

[Ál09] Alejandro Álvarez Ayllón, Antonio García Domínguez, Manuel Palomo Duarte y Inmaculada Medina Bulo. Los casos de prueba en la generación dinámica de invariantes en composiciones de servicios web con WS-BPEL. páginas 139–152. SISTEDES, septiembre 2009. ISBN 987-84-692-6832-2.

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section

when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses,

the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.