# Mecanismo de transformación de diagramas UML de casos de uso a código WSCDL

Isaac Machorro Cano[1], Daniel Eduardo Díaz Navarro[2], Giner Alor Hernández[3], Mónica Guadalupe Segura Ozuna[1], Lisbeth Rodríguez Mazahua[3]

[1] Universidad del Papaloapan (UNPA), Tuxtepec, Oaxaca, México

[2]Instituto de Estudios Superiores de Oaxaca A.C. (IESO), Oaxaca, Oaxaca, México

[3] Instituto Tecnológico de Orizaba (ITO), Orizaba, Veracruz, México

{imachorro, msegura}@unpa.edu.mx {dandiaznavarro}@gmail.com
{galor,lrodriguez}@itorizaba.edu.mx

**Resumen.** Los diagramas UML de Casos de Uso definen el comportamiento de un sistema a través de las interacciones de los actores y los procesos, además determinan los requisitos funcionales del sistema. Adicionalmente, las empresas están diseñando sus procesos de negocios mediante la tecnología de servicios Web y UML se ha posicionado como un lenguaje estándar de modelado para el desarrollo de aplicaciones. Por otra parte, con la utilización de las tecnologías de servicios Web y UML, es posible modelar nuevos procesos de negocios más complejos, los cuales requieren de mecanismos para lograr una mejor interoperabilidad a nivel empresarial. Estos mecanismos se logran a través de lenguajes composicionales en servicios Web como WS-CDL el cual se basa en XML. Este trabajo presenta un mecanismo de transformación que permite la generación de código WSCDL a partir de diagramas UML de Casos de Uso y un caso de estudio para ejemplificar su funcionalidad.

**Palabras clave:** mecanismo de transformación, WS-CDL, diagramas de casos de uso.

## 1. Introducción

Los diagramas UML (*Unified Modeling Language*) de Casos de Uso (DCU) permiten describir las funciones de un sistema desde el punto de vista de sus interacciones con él, sin detallar la descripción de la implementación y determinan los requisitos funcionales del sistema [1]. Los DCU son también la base para los diagramas relacionados de componente y despliegue. Dentro de los elementos que conforman los DCU se tienen: a) Actores.- Se le llama así al usuario, cuando desempeña ese papel con respecto al sistema. Estos llevan a cabo los casos de uso, un mismo actor realiza mu-

chos casos de uso; b) Caso de Uso.- Describe una tarea que realiza el sistema, representa también cómo el sistema colabora con el actor para realizar al menos una tarea; c) Relaciones entre Casos de Uso.- Entre los casos de uso se establecen tres tipos de relaciones: (1) Extends: es similar a un caso de uso, al aplicar esta relación implica que un caso de uso difiere en su comportamiento dependiendo de ciertas circunstancias. Incorpora implícitamente el comportamiento de otro caso en el lugar especificado indirectamente por este caso; (2) Include: incorpora explícitamente el comportamiento de otro caso en algún lugar de su secuencia; (3) Generalización: hereda el comportamiento y significado de otro caso de uso [2]. Así mismo el creciente desarrollo y utilización de Internet en el ámbito comercial, ha propiciado que este sea el lugar donde la mayoría de las organizaciones ponen a disposición sus modelos comerciales, ayudándose de los servicios Web. El W3C (*Consortium World Wide Web*), define a los servicios Web (SW) como un sistema diseñado para soportar la interacción máquina a máquina sobre una red que cuenta con una interfaz descrita en un formato que se procesa en una máquina WSDL (*Web Services Description Language*). La idea principal es que los servicios tradicionales se manejen como un SW, además los SW son sustancialmente un nivel de infraestructura existente entre modelos de componentes [3]. Para el desarrollo de estos SW existen muchas herramientas y lenguajes que facilitan la creación de dichos servicios, tales como XML (*eXtensible Markup Language*), SOAP (*Simple Object Access Protocol*), UDDI (*Universal Description, Discovery and Integration*) y WSDL [4].

Por otra parte, con la utilización de las tecnologías de SW y UML, es posible modelar nuevos procesos de negocios más complejos, los cuales requieren de mecanismos para lograr una mejor interoperabilidad a nivel empresarial. Estos mecanismos se logran a través de lenguajes composicionales en SW tales como WSCI (*Web Service Choreography Interface*), BPML (*Business Process Modeling Languaje*), WSFL (*Web Service Flow Languaje*), XLANG (*Web Services for Business Process Desing*) y los más utilizados como BPEL (*Business Process Execution Language for Web Services*) y WS-CDL (*Web Services Choreography Description Language*). WSCDL es mejor y más poderoso que BPEL, debido a que se centra básicamente en la descripción de la coreografía de los SW, además proporciona una definición de los formatos de información que se intercambia por todos los participantes, mientras que BPEL solo lo realiza para un participante. Así mismo, WSCDL proporciona el intercambio global de mensajes entre los participantes sin un punto de vista específico, mientras que BPEL lo realiza desde el punto de vista de solo un participante. WSCDL ofrece reglas que utiliza cada participante para estimar el estado de la coreografía y deducir cuál podría ser el siguiente intercambio de mensajes y BPEL especifica reglas que se ejecutan para deducir las actividades posteriores a realizar, una vez que se calcula la regla, el tiempo de ejecución de la orquestación ejecuta las actividades correspondientes. Además WSCDL ofrece un modelo para la especificación del mejor entendimiento del intercambio de mensajes y BPEL no tiene tal concepto. Tomando en cuenta lo descrito anteriomente, en este trabajo se presenta un mecanismo de transformación que permite la generación de código WSCDL a partir de diagramas UML de Casos de Uso y un caso de estudio para ejemplificar la funcionalidad del mecanismo de transformación propuesto; en donde se realiza un mapeo de DCU a las etique-

tas de WSCDL, se genera su correspondiente documento XMI (*XML Metadata Interchange*) y se desarrolla un conjunto de clases en Java para la implementación y generación de código WSCDL.

Este trabajo se estructura como sigue, en la sección 2 se presenta la estructura y descripción de un documento WSCDL, en la sección 3 se presenta el trabajo relacionado, en la sección 4 se presenta el mecanismo de transformación de DCU a código WSCDL, en la sección 5 se presenta el caso de estudio de la búsqueda del proveedor que ofrezca el precio más bajo de un libro y finalmente en la sección 6 se presentan las conclusiones, el trabajo a futuro y los agradecimientos.

## 2. WSCDL

WS-CDL describe colaboraciones interoperables entre sus participantes. Estas colaboraciones tienen un entorno común, donde hay una serie de reglas definidas y restricciones que se deben seguir para el intercambio de la información. Diferentes características se incluye en el lenguaje para proveer una reflexión completa y exacta de qué es necesario para modelar una coreografía en particular y dependiendo de lo que requieran, algunas características del lenguaje se determina si son completamente necesarias o no [5].

**Tabla 1.** Estructura de un documento WSCDL.

| Etiqueta WSCDL | Descripción |
|---|---|
| <package> | |
| name="NCName" | Nombre del documento WS-CDL |
| author="xsd:string" | Autor del documento WS-CDL |
| version="xsd:string" | |
| targetNamespace="uri" | |
| xmlns="http://www.w3.org/2005/10/cdl"> | |
| <informationType/> | Detalla el tipo de información de la variable o el token de referencia. |
| <token/> | Es un alias que se usa para hacer referencia a una variable o partes de ella. |
| <tokenLocator/> | Localiza un token dentro de un mensaje |
| <roleType/> | Refiere el comportamiento potencial que un participantType puede mostrar en una interacción |
| <relationshipType/> | representa las relaciones entre los roles |
| <participantType/> | Agrupa aquellas partes del comportamiento que debe de implementar por la misma entidad lógica. |
| <channelType/> | Representa un canal donde se realiza una colaboración entre "participantTypes" puntualizando donde sucede el intercambio de información. |
| Choreography-Notation | Este elemento especifica una colaboración entre "participantTypes que intercambian información dentro de la coreografía |
| <package | |

La estructura de un documento WS-CDL se muestra en la Tabla 1, en donde se observan los elementos a utilizar en una coreografía así como una breve descripción de cada uno de ellos. WS-CDL es una descripción y no un lenguaje ejecutable, se utiliza para describir colaboraciones de servicios notables; esto se entiende como un protoco-

lo comercial. Además WSCDL es un lenguaje basado en XML que se usa para describir la conducta colaborativa y común entre múltiples servicios que necesitan interactuar entre si en un orden para lograr alguna meta.

En la siguiente sección se presentan algunos trabajos relacionados con esta propuesta, en donde se hace uso de diagramas UML, WSCDL u otros lenguajes composicionales, SW y del lenguaje de programación Java entre otros aspectos relacionados con este trabajo.

## 3. Trabajos relacionados

En [6], se propone un nuevo enfoque basado en la planeación, incluyendo la compilación de contingencias, estado de acciones, análisis de dependencia y comunicación que convierten automáticamente una tarea de composición a una especificación distribuida de la coreografía. Por otra lado en [7] se propone un simple lenguaje llamado Chor c, el cual es un sub-idioma de interacción para modelar el aspecto del canal de la coreografía del servicio, además se estudian algoritmos para la comprobación estática de la coreografía del servicio y la generación del canal. Por otra parte en [8], se presentan cinco teoremas los cuales son útiles para trabajar con el modelado de diagramas de secuencia y establecer una teoría del modelo orientado a aspectos o un enfoque de la transformación de los modelos. En [9] se extraen formalmente especificaciones basadas en WSCDL en un proceso de comunicación secuencial soportando un enfoque formal para comprobar los modelos de SW, esto a través de un ejemplo PAT adaptado mediante la realización de un mapeo de WSCDL a CSP con el objetivo de obtener las descripciones de CSP de la coreografía de SW. Así mismo en [10], se presenta una revisión de trabajos sobre la integración de las representaciones de procesos de negocios y la tecnología de SW, identificando la necesidad de desarrollar propuestas de transformación entre modelos, representar mapeos de una manera formalizada y una estructura de ejecución común.

Por otra parte en [11], se presenta una metodología para el desarrollo de los SW basados en diagramas de secuencia en UML 2.0 como especificaciones de partida en la composición de SW, estos diagramas se traducen en lenguaje WS-CDL. Además en [12], se propone un método basado en CPN que detecta los errores en la fase de diseño y asegura la exactitud de la composición de servicios para un análisis estructural estático y un análisis de comportamiento dinámico, presentando un ejemplo del mapeo entre los elementos de la composición de SW y CPN. Así mismo en [13], se presenta la herramienta WSCVT, la cual utiliza Java para verificar un algoritmo RBSR, en donde el enfoque propuesto se verifica utilizando la herramienta WSCVT y los resultados revelaron un mejor rendimiento en términos de evitar problemas de escasez y accesibilidad. Además se presenta una traducción de los términos de WSCDL en relación a los términos de un RBA. Por otra parte en [14], se presenta una verificación de los SW compuestos utilizando la interface de autómatas, en donde inicialmente la composición se realiza mediante el uso de BPEL, el cual se convierte en autómatas de interfaz dentro de Promela y posteriormente verificadas utilizando la herramienta llamada SPIN. Además en [15], se presenta una construcción de SW utilizando Re y

autómatas restringidos; igualmente proponen una estructura que tomando como entrada la descripción del comportamiento del servicio, sus interfaces WSDL y su interacción en Reo, genera el correspondiente código en Java para orquestar los servicios.

En la siguiente sección se presenta el mecanismo de transformación de DCU a código WSCDL en donde se describe un mapeo de un DCU a sus correspondientes etiquetas de WSCDL, la descripción de las etiquetas de la coreografía del código en WSCDL y la descripción de las clases en Java desarrolladas.

## 4. Mecanismo de transformación

Para el mecanismo de transformación propuesto en este trabajo que consiste en la transformación de DCU a su correspondiente código en WS-CDL, previamente se han realizado otras pruebas con los diagramas UML de Actividades y de Secuencia, realizando un mapeo de cada diagrama a las etiquetas en WSCDL y generando a partir de los diagramas su correspondiente código en WSCDL, mediante diversos escenarios presentados [16]. Por lo tanto, en esta propuesta se utilizó un DCU que representa el caso de estudio de la "Búsqueda del proveedor que ofrezca el precio más bajo de un libro". La Fig. 1 muestra el mapeo de las etiquetas correspondientes al lenguaje WS-CDL y su correspondiente descripción relacionada al DCU en donde:

1. La etiqueta <relationshipType> se genera principalmente al pasar de un rol a otro. Por ejemplo al pasar del rol WS-CDL al rol BPEL4WS.
2. La etiqueta <informationType> y <variables> se generan con la información que se va obteniendo de cada caso de uso, como por ejemplo "Ingresar dato del libro", "Buscar por autor", entre otros.
3. La etiqueta <exchange> se genera al haber intercambio de información entre el caso de uso "Buscar Libro", que está dentro del rol WS-CDL al caso de uso "Buscar precio y proveedor del libro" del rol BPEL4WS.
4. La etiqueta <token> y <tokenLocator> se crean cuando se localizan y definen los alias a los casos de uso, esto es la información de cada uno, por ejemplo el caso de uso "Buscar libro".
5. La etiqueta <roleType> se genera cuando existe por lo menos un rol dentro del DCU, en este caso se cuentan con tres roles: WS-CDL, BPEL4WS y WSDL.
6. La etiqueta <interation> se obtiene cuando se pasa de un caso de uso a otro.

Finalmente en la etiqueta <choreography> se agrupan todas las etiquetas anteriormente mencionadas e incluimos la etiqueta <sequence> que representa la secuencia de intercambio de información entre los casos de uso, esto para realizar su correspondiente coreografía en WSCDL.

Una vez identificadas las etiquetas de WSCDL en el diagrama, el código generado en WSCDL se muestra en la Fig. 2, en donde se observa que:

1. El inicio de la coreografía es mediante la etiqueta <choreography> en donde se indica el nombre y descripción del caso de estudio.

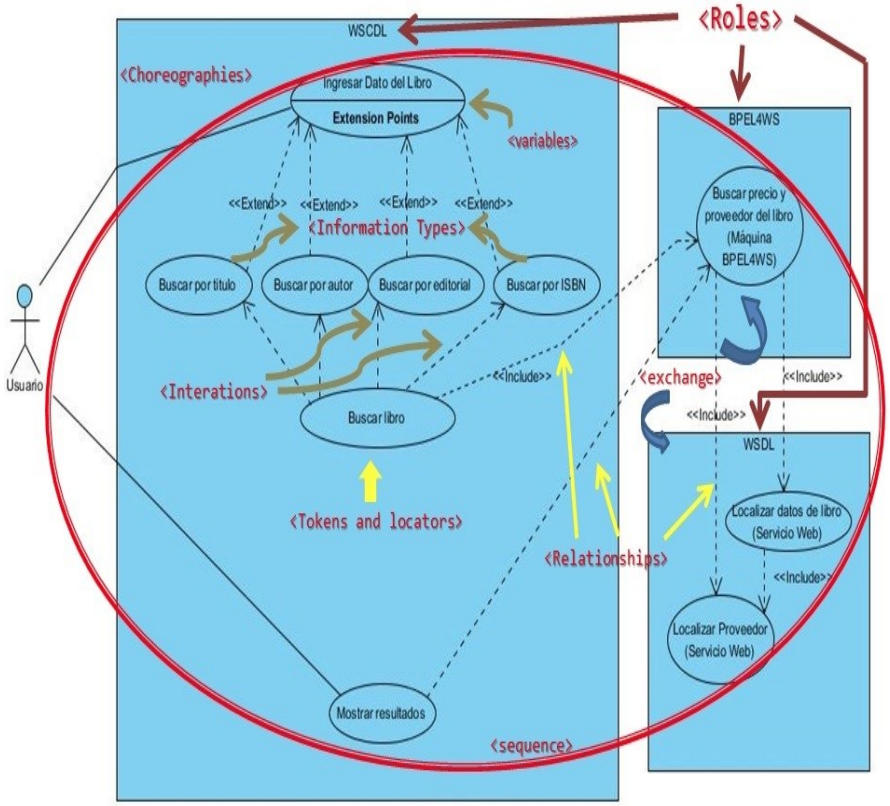


**Fig. 1** Diagrama UML de Casos de Uso

2. Posteriormente en la etiqueta <variableDefinitions> se indican todas las variables a utilizar obtenidas de los diferentes casos de uso, indicando en cada una de ellas para que se utilizará.
3. Además a través de la etiqueta <sequence>, se indica que la coreografía es secuencial. Después indicamos con la etiqueta <iteraction> el tipo de caso de uso, dentro del cual se establece la operación y su correspondiente canal.
4. Por otra parte en las etiquetas <participate> <relationshipType> se indica el tipo de relación entre los participantes de la coreografía a través de los roles, para este caso los roles son “WSCDL”, “BPEL4WS” y “WSDL”.
5. Para la comunicación entre los diferentes casos de usos se utiliza la etiqueta <exchange>, en donde para realizar el envío de mensajes se realiza mediante la etiqueta <send> y la recepción de los mismos con la etiqueta <receive>, así mismo el tipo de acción que se desea realizar a través de las palabras reservadas “response” o “request” según sea el caso.

Finalmente para el cierre de cada etiqueta abierta se antepone el símbolo “/” según sea el caso; la coreografía del caso de estudio propuesto finaliza con la etiqueta </choreography>.

```
<choreography
name="B\u00fasqueda del proveedor que ofrezca el precio m\u00e1s bajo de un
libroChoreography" root="true">
<descriptiontype="documentation">TheChoreographyfor B\u00fasqueda del proveedor
que ofrezca el precio m\u00e1s bajo de un libro</description>
<relationship type="tns:BPEL4WS2WSCDL"/>
<variableDefinitions>
<variable channelType="tns:BPEL4WS2WSCDLChannel"
                name="BPEL4WS2WSCDLC" roleTypes="tns:BPEL4WSRole tns:WSCDLRole">
<description type="documentation">Channel Variable</description>
</variable>
<variable informationType="tns:BuscarlibroType"
                name="Buscarlibro" roleTypes="tns:BPEL4WSRole tns:WSCDLRole">
<description type="documentation">Request Message</description>
</variable>
<variable
informationType="tns:Buscar precio y proveedor del libro (M\u00e1quina
BPEL4WS)Type"
name="Buscar precio y proveedor del libro (M\u00e1quina BPEL4WS)"
roleTypes="tns:BPEL4WSRole tns:WSCDLRole">
<description type="documentation">Response Message</description>
</variable>
</variableDefinitions>
<sequence>
<interaction channelVariable="tns:BPEL4WS2WSCDLC"
                name="Interaction1" operation="Operation1">
<description type="documentation">Interaction1</description>
<participate fromRoleTypeRef="tns:BPEL4WSRole"
relationshipType="tns:BPEL4WS2WSCDL" toRoleTypeRef="tns:WSCDLRole"/>
<exchange action="request"
informationType="tns:BuscarlibroType" name="Buscarlibro">
<description type="documentation">Request Message Exchange</description>
<send variable="cdl:getVariable('Buscarlibro','','')"/>
<receive variable="cdl:getVariable('Buscarlibro','','')"/>
</exchange>
<exchange action="respond"
informationType="tns:Buscar precio y proveedor del libro (M\u00e1quina
BPEL4WS)Type" name="Buscar precio y proveedor del libro (M\u00e1quina BPEL4WS)">
<description type="documentation">Response Message Exchange</description>
<send variable="cdl:getVariable('Buscar precio y proveedor del libro
(M\u00e1quina BPEL4WS)','','')"/>
<receive variable="cdl:getVariable('Buscar precio y proveedor del libro
(M\u00e1quina BPEL4WS)','','')"/>
</exchange>
</interaction>
</sequence>
</choreography>
```

**Fig. 1.** Código WSCDL

La implementación del mecanismo de transformación se llevó a cabo siguiendo los tres pasos que a continuación se describen:

**Paso 1.-** Se modeló el DCU del caso de estudio de la búsqueda del proveedor que ofrezca el precio más bajo de un libro en la herramienta de modelado UML *Visual Paradigm for UML 8.0 Enterprise Edition* la cual es una herramienta que colabora en el diseño de software.

**Paso 2.-** Una vez generado el diagrama con la herramienta de modelado, se exportó el DCU a un documento XMI, el cual es un estándar de la OMG (*Object Management Group*) para el intercambio de información de metadatos utilizando XML. En la Tabla 2, se indica la relación de los elementos de un documento XMI con respecto a las etiquetas del código en WSCDL.

**Tabla 2.** Relación entre XMI y WSCDL

| Documento XMI | Código WSCDL | |
|---|---|---|
| <uml:Diagram> | <package> | |
| <ownedMember> | roleType> | |
| <uml:Model> | <participantType> | <choreography> |
| | <relashionsipType> | <variableDefinitions> |
| <include> | <exchange> | <sequence> |
| **addition** (origen) | <send> | <interaction> |
| | response<br>request | <participate><br><relationshipType><br><exchange> |
| **xmi:id** (destino) | <recibe> | <send> |
| | response<br>request | response<br>request |
| <extends> | <exchange> | <recibe> |
| **extendedCase** (origen) | <send> | response<br>request |
| | response<br>request | </exchange> |
| **xmi:id** (destino) | <recibe> | . |
| | response<br>request | .<br>. |
| <uml:Dependency> | <exchange> | <exchange> |
| **client** (origen) | <send> | <send> |
| | response<br>request | response<br>request |
| **supplier** (destino) | <recibe> | <recibe> |
| | response<br>request | response<br>request |
| <ownedMember> | <informationType> | </exchange> |
| | <token><br><tokenLocator | </choreography> |

**Paso 3.-** Para la generación del código se desarrollaron cuatro clases en el lenguaje de programación Java, el cual cuenta con JAXP (*Java API for XML Processing*) una API (*Application Programming Interface*) que sirve para procesar archivos XML. JAXP aprovecha los estándares de las API´s SAX (*Simple API for XML*) y DOM (*Document Object Model*), de tal manera que se analizan los datos como secuencia de eventos, de igual forma, al ser diseñado para ser flexible, JAXP permite usar cualquier analizador que trabaje con XML. SAX igualmente permite interpretar un archivo que utiliza XML detectando cuando empieza y termina un elemento del documento, además de comprobar que el documento este bien formateado. SAX no carga todo el documento en memoria solo lo que se está analizando, por lo cual es muy eficiente en cuanto al tiempo y memoria que se usan en el análisis. Así mismo para generar el código WSCDL, se utilizó la API DOM la cual describe la estructura para un documento XML, con ella podemos navegar en el documento e insertar nodos. La especificación que hace el W3C define a DOM como una interfaz independiente de cualquier plataforma que permite a programas acceder y actualizar dinámicamente la estructura del documento. En la Figura 3 se muestran las cuatro clases que se desarrollaron para la realización de este proyecto, en donde:

1. La **clase FramePrincipal**.- Dibuja un Frame para seleccionar el documento XMI, invoca a la clase "CasoDeUso" y visualiza el documento WSCDL.
2. La **clase CasoDeUso**.- Invoca a las clases "ObtenerEtiquetas" y "Generar-WSCDL".
3. En la **clase ObtenerEtiquetas**.- Se definen los métodos para obtener los elementos del código WS-CDL, a través del archivo XMI. Esta clase cuenta los métodos: getNameDiagram( ) obtiene el nombre del diagrama del archivo XMI, getAuthor-Diagram( ) obtiene el autor del diagrama de casos de uso, getRole( ) extrae los diferentes grupos que interactúan dentro del diagrama, getCasosDeUso( ) obtiene la información de los casos de uso que se encuentran dentro de cada grupo, getRelations( ) obtiene las relaciones que existen entre los casos de uso y el método getActionCasos( ) extrae que acción representa cada caso de uso, ya sea un "request" o un "response" hacia otro caso de uso.
4. La **clase GenerarWSCDL**.- Cuenta con el método createWSCDL ( ) el cual genera el documento WS-CDL con la información obtenida en la clase "ObtenerEtiquetas" del archivo XMI.

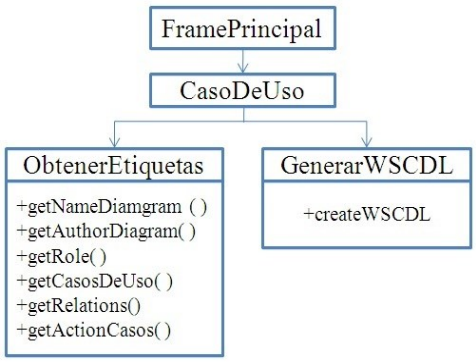


**Fig. 2.** Clases en Java

En este mecanismo de transformación aplicado al caso de estudio propuesto, se observa la generación de código WSCDL a partir de un diagrama UML de Casos de Uso, mostrando el mapeo del diagrama UML de Casos de Uso a los elementos del código en WSCDL, la relación entre el documento XMI con el código WSCDL, así como su correspondiente implementación.

En la siguiente sección se presenta un caso de estudio de la búsqueda del proveedor que ofrezca el precio más bajo de un libro en el cual se basa este mecanismo de transformación y que sirvió para la validación del mismo.

## 5 Caso de estudio: "Búsqueda del proveedor que ofrezca el precio más bajo de un libro"

Para la validación del mecanismo de transformación de DCU a código WSCDL se planteó el siguiente escenario: Un usuario desea saber el proveedor que ofrece el pre-

cio más bajo de un determinado libro. Para esto, recurre a un servicio que le oriente en su búsqueda y le proporcione diversos resultados. De los proveedores propuestos, el usuario decidirá cuál de ellos es el que ofrece el precio más bajo del libro.

Una vez planteado el caso de estudio, se realizó el modelado del correspondiente DCU en donde como se muestra en la Fig. 1 se identifican los tipos de relaciones, los diferentes Casos de Uso y a qué rol pertenecen. Además se inicia con el caso de uso "Ingresar Dato del Libro" en donde se realiza la búsqueda del mismo a través de los casos de uso "Buscar por título", "Buscar por autor", "Buscar por editorial" o "Buscar por ISBN " según sea el caso para pasar al caso de uso "Buscar libro" ubicados en el sistema WSCDL. El caso de uso "Buscar libro" se comunica con el caso de uso "Buscar precio y proveedor del libro (Máquina BPEL4WS)" ubicado en el sistema BPEL4WS en donde este realiza la petición de los datos correspondientes a los casos de uso "Localizar Proveedor (Servicios Web)" y "Localizar datos del libro (Servicio Web)" pertenecientes al sistema WSDL. Posteriormente se envía dicha información al caso de uso "Buscar precio y proveedor del libro (Máquina BPEL4WS)" para finalmente presentar los resultados de la búsqueda al usuario a través del caso de uso "Mostrar resultados" para que él tome la decisión de la compra del libro de acuerdo al proveedor que ofrezca el precio más bajo.

Después de modelado y descrito el funcionamiento del caso de estudió en el DCU, generamos a partir del diagrama el correspondiente archivo XMI, del cual obtenemos la representación de los elementos del lenguaje WS-CDL como se describe a continuación:

- Del documento XMI obtenemos el contenido de la etiqueta <uml:Diagram> que representa la etiqueta <package> en el documento WS-CDL.
- Dentro de la etiqueta <ownedMember> en el documento XMI, se encuentra la etiqueta <uml:Model> la cual representa la etiqueta "roleType" en WSCDL de donde se obtiene el elemento "participantType", además de la etiqueta "relashionsipType" ya que esta se da cuando dos roleType se comunican entre sí.
- Para obtener el elemento informationType en WSCDL, se busca la etiqueta <ownedMember> en el documento XMI, el cual debe de estar dentro de otra etiqueta con el mismo nombre, a partir de este elemento podemos extraer los elementos token y tokenLocator de WSCDL.
- El elemento choreography en WSCDL, se obtiene a partir de los elementos anteriormente mencionados asi mismo se genera el elemento <interaction> en donde si existe un intercambio de información entre dos roleType se crea también la etiqueta <exchange>. En el documento XMI este intercambio lo obtenemos de diferentes etiquetas, una es <include> con su elemento "addition" que representa el origen y "xmi:id" que es el destino, de igual forma con la etiqueta <extends> y su atributo "extendedCase" que es el origen y el destino es "xmi:id", y la última forma de obtener la etiqueta <exchange> es con la etiqueta <uml:Dependency> dentro del XMI, siendo el origen el atributo "client" y el destino el atributo "supplier".

Una vez identificados y relacionadas las etiquetas del documento XMI con las etiquetas de WSCDL, la clase "ObtenerEtiquetas" lee el archivo XMI y guarda la información descrita anteriormente a través sus métodos para posteriormente pasar a la

clase “GenerarWSCDL" donde con dicha información se procede a generar el código en WS-CDL. Finalmente, se obtiene el código WS-CDL el cual es visualizado en cualquier editor o navegador que soporte XML.

En la siguiente sección se presentan las conclusiones a la que se llegaron al desarrollar el mecanismo de transformación, fundamentalmente al realizar el mapeo y al generar las clases en Java para la generación de código WSCDL, se plantea el trabajo a futuro y se presentan los agradecimientos.

## 6. Conclusiones y trabajo a futuro

Los SW son una plataforma para mostrar la interoperabilidad de servicios de las aplicaciones utilizando como estándar UML, el cual también se utiliza para la descripción de procesos de negocios. En este contexto los diagramas UML de Casos de Uso describen el comportamiento de un sistema visto desde el usuario, además estos diagramas determinan los requisitos funcionales del sistema y son importantes para la visualización, especificación, documentación del modelo estructural y construcción de un sistema. Además describen de una forma gráfica la composición de un SW, por lo cual es más fácil el entendimiento de su funcionalidad. Por otra parte podemos describir al lenguaje WS-CDL como punto de partida para la generación de la coreografía de los SW, mostrando la secuencia de mensajes entre las distintas partes y fuentes, gracias a que este lenguaje se basa en XML. En este trabajo se presentó un mecanismo de transformación que permite la generación de código WSCDL a partir de diagramas UML de Casos de Uso basado en el caso de estudio de la búsqueda del proveedor que ofrezca el precio más bajo de un libro lo cual permitió validar el mecanismo de transformación presentado.

Finalmente el trabajo a futuro consiste en analizar otros casos de estudio para validar y fortalecer el mecanismo de transformación de DCU a código WSCDL propuesto. Además se considera desarrollar adicionalmente bibliotecas de funciones en Java para generar código WSCDL a partir de diagramas UML de componentes y despliegue como continuación de este trabajo.

**Agradecimientos.** Los autores agradecen al Programa de Fortalecimiento de la Calidad en Instituciones Educativas (PROFOCIE 2014), al Programa para el Desarrollo Profesional Docente (PRODEP), al Consejo Nacional de Ciencia y Tecnología (CONACYT) y al Tecnológico Nacional de México por el apoyo otorgado para la realización de esta investigación.

## Referencias

1. Cook, S.: Looking back at UML. Software & Systems Modeling, Vol. 11, Issue 4, pp. 471–80, Springer-Verlag (2012)
2. Falgueras, C. B.: Ingeniería de software. pp. 71–107, Editorial UOC (2003)
3. Bittner, K., Spence, L.: Use Case Modeling. pp. 3–18, Pearson Education (2003)

4. Arboleda, C. L. M.: Servicios WEB: Distribución e integración. Vol. 2, S & T Sistemas y Telemática (2004)
5. Ross, T. S., Fletcher, T.: Web Services Choreography Description Language: Primer W3C Homepage, http://www.w3.org/TR/2006/WD-ws-cdl-10-primer-20060619/ (2014)
6. Zou, G., Gan, Y., Chen, Y., Zhang, B., Huang, R., Xu, Y., Xiang, Y.: Towards automated choreography of Web services using planning in large scale service repositories. Applied Intelligence, Vol. 41, Issue 2, pp. 383–404, Springer US (2014)
7. Yang, H., Cai, C., Peng, L., Zhao, X., Qiu, Z., Qin, S.: Algorithms for checking channel passing in web services choreography. Frontiers of Computer Science, Vol. 7, Issue 5, pp. 710–728, Springer Berlin Heidelberg (2013)
8. Gronmo, R., Kobro, R. R., Moller, P. B.: Confluence of aspects for sequence diagrams. Software & System Modeling, Volume 12, Issue 4, pp. 789–824, Springer Berlin, Heidelberg (2011)
9. Dong, X., Zhou, L., Wei-min, L., Bo-feng, Z.: Model checking web services choreography in process analysis toolkit. Journal of Shanghai University, Vol. 14, Issue 1, pp. 45–49, Shanghai University and Springer-Verlag (2010)
10. Grolinger, K., Capretz, M. A. M., Cunha, A., Tazi, S.: Integration of business process modeling and Web services: a survey. Service Oriented Computing and Applications, Vol. 8, Issue 2, pp. 105–128, Springer (2013)
11. Cambronero, M. E., Ruiz, V. V., Martínez, E.: Design and Generation of Web Services Choreographies with Time Constraints. J. UCS, Vol. 17, Issue 13, pp. 1800–1829 (2011)
12. Tian, B., Gu, Y., Formal Modeling and Verification for Web Service Composition. Journal of software, Vol. 8, No. 11, Academy publisher (2013)
13. Danapaquiame, N., Ilavarasan, E.: Corroboration Strategy for Web Services Choreography using Revise Buchi Automata. International Journal of u- and e- Service, Sciencie and Technology, Vol. 6, No. 5, pp 115–132 (2013)
14. Mei, J., Miao, H., Chen, Y., Gao, H.: Verifying Web Service composition based on Interface Automata using SPIN. Journal of Digital Content Technology and it's applications, Vol. 4, No. 8 (2011)
15. Jongmans, S. T. Q., Santini, F., Sargolzaei, M., Arbab, F., Afsarmanesh, H.: Orchestrating web services using Reo: from circuits and behaviors to automatically generated code. Service Oriented Computing and Applications, Vol. 8, Issue 4, pp. 277–297, Springer (2013)
16. Alor Hernandez, G., Machorro Cano, I., Gomez, J. M., Cruz Ahuactzi, J., Posada Gomez, R., Mencke, M., Juarez Martinez, U.: Mapping UML Diagrams for generating WS-CDL code. In: Third International Conference on Digital Society (ICDS), IEEE (2009)