WS-CDL

Software development based on components and services

M.I. Capel

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

December, 18th 2020

Máster Universitario en Ingeniería Informática

Fundamentación

Web Service Choreography Description Language

- Se trata de una recomendación (*especificación candidata*) de W3C, 2005
- Describe un lenguaje que sirve para caracterizar las interacciones entre diferentes servicios Web (SW)
- Existen muy pocas herramientas software que permitan comprobar y ejecutar las especificaciones en el nuevo lenguaje WS-CDL
- Muchas características fundamentales de este lenguaje no están completamente claras en la actualidad

Definición

"The Web Services Choreography Description Language (WS-CDL) is an XML-based language that describes peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal", 2005 W3C(*c*) (MIT, ERCIM, Keio).

Importante

"In Web Services Choreography Description Language (WS-CDL) there is no single point of control. There are no global variables, conditions or workunits.
To have them would require centralised storage and orchestration.
WS-CDL does permit a shorthand notation to enable variables and conditions to exist in multiple places, but this is syntactic sugar to avoid repetitive definitions.
There is also an ability for variables residing in one service to be aligned (synchronized) with the variables residing in another service, giving the illusion of global or shared state."
(W3C Working Draft, 2005)

Importante

"In WS-CDL all messages are described as information types and have no special significance over each other.
All that WS-CDL describes is the ordering rules for the messages which dictate the order in which they should be observed.
When these ordering rules are broken WS-CDL considers them to be out-of-sequence messages and this can be viewed as an error in conformance of the services that gave rise to them against the WS-CDL description".
(W3C Working Draft, 2005)

¿Por qué utilizar WS-CDL?

Para asegurar:

- la interoperabilidad
- que el coste total máximo de los sistemas software es el más bajo posible mediante el aseguramiento de que los servicios que participan en el coreografía se comportan bien continuamente
- menor tiempo de pruebas y, de esta forma, se reduce el coste de la entrega del producto software

Introducción a WS-CDL

- A diferencia de WS-BPEL, CDL no describe comunicaciones entre 2 partes desde el punto de vista de uno de los participantes
- CDL describe escenarios de interacción que implican a múltiples partes y desde un punto de vista independiente de los procesos que participan, llamamos *modelo global* a una descripción WS-CDL de un conjunto de servicios

Introducción a WS-CDL-II

- El modelo global asegura la independencia del comportamiento concreto de cualquiera de los servicios componentes
  - WS-CDL no está ligado a WSDL ni a ningún otro lenguaje de descripción de servicios
  - WS-CDL puede jugar el mismo papel de modelo global tanto para servicios SOA como para SW
  - Se podría utilizar para describir un modelo global de los servicios sin utilizar descripciones-WSDL (sólo utilizando interfaces de Java, quizás); aunque tal modelo puede ser dependiente de la implementación
- Se necesitan especificaciones formales de la semántica del lenguaje que, en un futuro, permitan llevar a cabo la verificación de las descripciones escritas en WS-CDL

Introducción - III

- Comportamiento observable de un servicio; puede ser descrito utilizando BPEL abstracto y WSDL para describir secuencias de llamadas a funciones
- Concepto de *comportamiento observable colaborativo* común
- El comportamiento observable colaborativo no puede entenderse sólo como el conjunto de los comportamientos de todos los servicios operando juntos

Introducción - IV

- Las interacciones de SW se describen sin recurrir a la coordinación o recurrir a algún servicio coordinado
- Los servicios se sincronizan por el conocimiento mutuo derivado de una interacción de negocio
- Los lenguajes de coreografía describen los puntos de interacción entre los SW existentes sin necesidad de definir ningún nuevo servicio (como hace WS-BPEL)
- Muy adecuado para la descripción de procesos concurrentes y escenarios de interconexión dinámicos
- Basado en XML, carece de una representación gráfica y es útil como herramienta abstracta para definir interacciones entre SW

Introducción – V

Mecanismos para descripción de comportamiento común

- Alineamiento de información específica de un negocio (registro de aceptación de una orden de venta entre vendedor y comprador)
- Interacción (cuando un comprador pide precio a un vendedor)
- Declaración de interés en el progreso de una coreografía (ha terminado/comenzado la coreografía de comercio entre vendedor/comprador)

Introducción – VI

- Introduce nuevos conceptos de diseño, como la posibilidad de *paso de canales* entre procesos, y otras construcciones para expresar directamente la ejecución concurrente y la selección no–determinística, coreografías, etc.
- No existe todavía ninguna semántica formal de WS-CDL comúnmente aceptada que muestre de qué manera exactamente está relacionado este lenguaje con el π-Calculus
- Existen pocas herramientas que permitan experimentar con el nuevo lenguaje WS-CDL, la más conocida es `pi4soa`, que proporciona un plugin para Eclipse

WS-CDL Estático

La parte estática de WS-CDL define las entidades colaborantes dentro de una especificación escrita en este lenguaje.

Entidades estáticas

1. Tipos *rol*
2. Tipos *participante*
3. Tipos *relación*
4. Tipos *canal*

Tipos Rol

- Describen un comportamiento observable, es decir, como especificar uno o más *behavior*s, que exhibe un *participante* ; son equivalentes a una descripción WSDL
- El tipo *behavior* especifica las operaciones ofrecidas, que podrían ser definidas en un tipo-interfaz de WSDL
- No se pueden crear instancias a partir de una definición de uno de estos tipos *rol*

```
1 roleType ::=
2 <roleType name="ncname">
3         <description type="documentation" </description>?
4 <behavior name="ncname" interface="qname"? />+
5 </roleType>
```

Tipos Rol

Ejemplos de roles

```
<role Type name="Consumidor">
<behavior name="consumidorParaMinorista"
          interface ="rns:ConsumidorMinoristaPT"/>
<behavior name="consumidorParaAlmacenes"
          interface ="rns:ConsumidorAlmacenesPT"/>
</roleType>
<roleType name="Minorista">
<behavior name="minoristaParaConsumidor"
          interface="rns:MinoristaConsumidorPT"/>
</roleType>
```

Tipos Participante

- Define un conjunto de uno o más roles que desempeña un participante que colabora

Ejemplos de participantes

```
<participant Type name="Consumidor">
<roleType typeRef="tns:Consumidor"/>
</participantType>
<participantType name="Minorista">
<roleType typeRef="tns:Minorista"/>
</participantType>
```

Tipos Participante–II

- Por ejemplo, para agrupar los roles *Minorista* y *Almacén* en una interacción de negocios que son implementados por la misma organización

```
<participant name="Participante">
<roleType typeRef="tns:Consumidor"/>
<roleType typeRef="tns:Minorista"/>
</participant>
```

Tipos Relación

- Especifican los compromisos mutuos que los *roles/behaviors* de 2 participantes que colaboran han de satisfacer
- Definen el *enlace estático entre roles*, cómo están conectados los roles con el objetivo de interaccionar

```
relationshipType ::=
<relationshipType name="ncname">
<role type="qname" behavior="listaDeNcname"?/>
 <role type="qname" behavior="listaDeNcname"?/>
</relationshipType>
```

Tipos Relación – II

Ejemplos de relaciones

```
<relationshipType name="RelacionConsumidorMinorista">
<roleType typeRef="tns:Consumidor"
behavior="consumidorParaMinorista"/>
<roleType typeRef="tns:Minorista"
behavior="minoristaParaConsumidor"/>
</relationshipType>
```

Tipos Información

- Se utilizan para describir los tipos de las muchas variables que pueden utilizarse en una coreografía
- Para describir los tipos de mensajes que se pueden enviar entre los roles de una interacción

```
informationType ::=
<informationType name="ncname"
    type="qname"?|element="qname"?
    exceptionType="true"|"false"?/>
```

Resumen elementos fundamentales de WS-CDL

- Tipo Información:Este elemento identifica el tipo de información utilizada dentro de un coreografía para evitar hacer referencia directamente a tipos de datos en un esquema XML o un documento WS-CDL
- Tipo RoI: enumera los posibles comportamientos observables que un el participante puede exhibir para interactuar con otros
- Tipo Relación: describe la relación entre dos participantes para colaborar con éxito

Equivalencia entre los elementos de WS-CDL y un modelo de eventos

WS-CDL elementos	ModeloEventos
informationType, participantType, channelType	Set
roleType	Constant
relationshipType	Axiom

Tipos Token y Locator

- Un token nos proporciona un tipo y un alias de una información dentro de un documento,
  - aseguran que la comunicación entre canales “casa”
  - para referirse a una referencia de un servicio (URL)
  - hacen más legible y mantenible una información a través de la indirección que ellos soportan
- Un tipo *Token Locator* especifica reglas para poder seleccionar una información en un documento

```
1 <token   name="ncname"
2     informationType="qname"/>
3 <tokenLocator   tokenName="qname"
4     informationType="qname"
5     part="ncname"?
6     query="XPath-expression"/>
```

Tipos Canal

- Son el principal mecanismo *dinámico* de colaboración, usado por los participantes en una interacción
- Las instancias de tipos-canal se utilizan para indicar a dónde y cómo comunicar un mensaje:
  - Especifican el *rol/behavior* y la referencia de un participante que colabora con otros
  - Identifican instancias de *roles*
  - Identifican una instancia de una conversación entre 2 ó más participantes
- Los canales son nombrados, descritos y relacionados con los roles que realizan su interfaz comportamental

Tipos Canal - II

```
channelType::=
<channelType  name="ncname"
     usage="once"|"unlimited"?
     action="request-respond"|"request"|"respond"? >
<passing  channel="qname"
      action="request-respond"|"request"|"respond"?
     new="true"|"false"? />*
<role  type="qname"  behavior="ncname"? />
<reference>
<token name="qname"/>
</reference>
<identity>
<token name="qname"/>+
</identity>?
</channelType>
```

Tipos Canal - III

Ejemplos de canales

```
<channelType name="CanalConsumidor">
<roleType typeRef="tns:Consumidor"/>
<reference><token name="tns:consumidorRef"/></reference>
<identity><token name="tns:ordenCompraID"/></identity>
</channelType>
<channelType name="CanalMinorista">
<passingchannel="CanalConsumidor" action="request"/>
<roleType typeRef="tns:Minorista"
behavior="minoristaParaConsumidor"/>
<reference><token name="tns:minoristaRef"/>
</reference>
<identity><token name="tns:ordenCompraID"/></identity>
</channelType>
```

Tipos Canal–IV

- Las comunicaciones en WS-CDL siempre conectan 2 partes, ya que las comunicaciones múltiples no son posibles
- La interacción a la que dar lugar una comunicación posee 2 fases: (1) petición, (2) respuesta (opcional)
- El tipo canal identifica el tipo de rol del que responde pero no el tipo de rol de la parte que pregunta
- El tipo canal puede incluir restricciones en el uso de las instancias
- Contienen definiciones de tipos y de funciones *XPath*, para recuperar datos desde un documento basado en XML

Tipos Canal–V

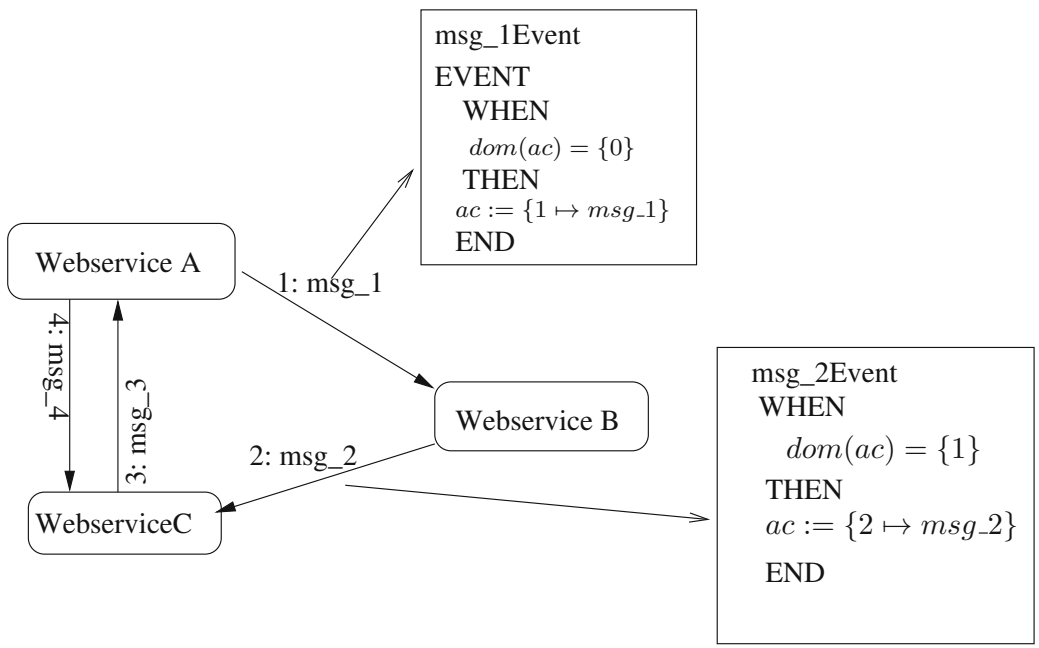
- Un rol puede pasar 1 ó más canales a otro o más roles
- Las canales se pueden pasar como el resultado de una selección no–determinista, creándose de esta manera nuevos puntos de colaboración dinámicamente
- Un *tipo canal* puede restringir los tipos de canales que se pueden intercambiar a través de ese canal entre los servicios web participantes
- Un tipo canal puede restringir su utilización especificando el número de veces que un canal puede ser usado

Ejemplo inicial

Enunciado

" Se trata de coreografiar el mercadeo entre un comprador y un vendedor de forma simplificada. El comprador simplemente pide un precio al vendedor y el vendedor responde con un precio o una excepción si no conoce los artículos o no están disponibles ahora"
(W3C, Definición WS-CDL, 2005)

Transformación de las interacciones de coreografias CDL en un modelo de eventos



Ejemplo inicial–diagrama

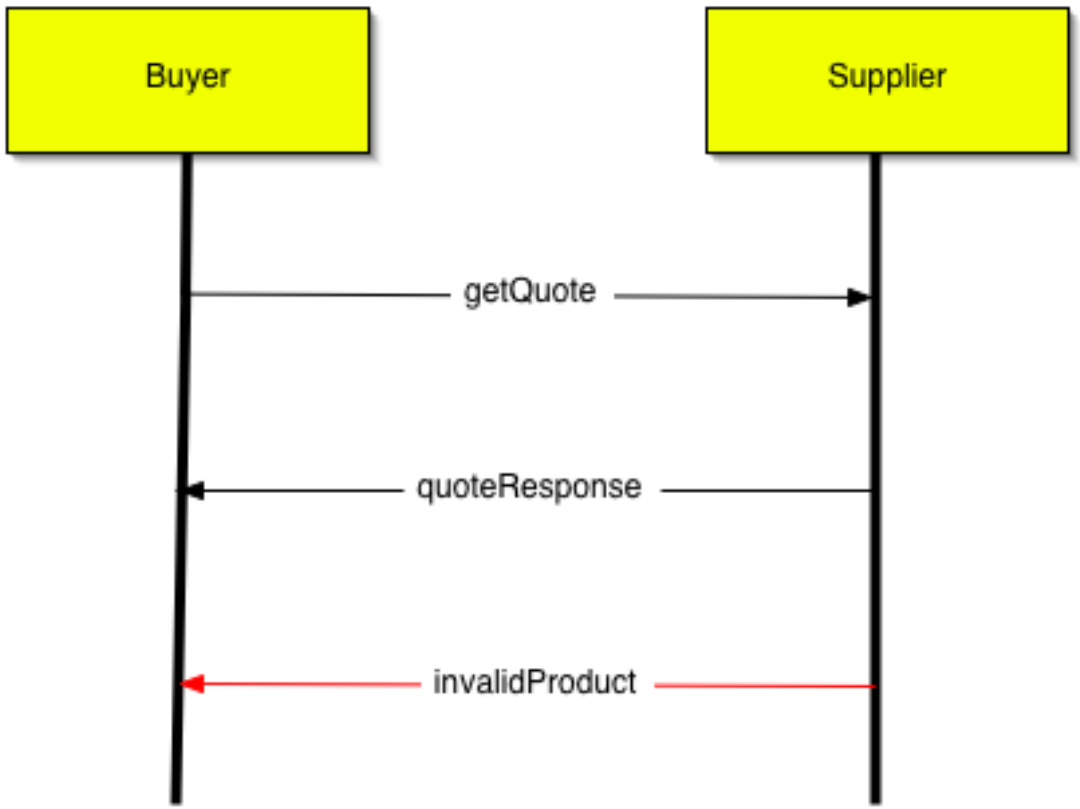


Figura: Ejemplo inicial para descripción WS-CDL

fragmento de WS-CDL

```
<interaction name="ObtencionPrecio" operation="
    conseguirPresupuesto" channelVariable="tns:
    CompradorParaVendedorC">
    <description type="documentation">
        Obtencion Presupuesto  </description>
    <participate relationshipType="tns:CompradorParaVendedor"
        fromRoleTypeRef="tns:RolComprador" toRoleTypeRef="tns:
        RolVendedor"/>

    <exchange name="PeticionPresupuesto" informationType="tns:
        TipoPeticionPresupuesto" action="request">
        <description type="documentation">
                Intercambio Mensaje Peticion Presupuesto
        </description>
        <send variable="cdl:getVariable('peticionPrespuesto
            ','','')"/>
        <receive variable="cdl:getVariable('peticionPrespuesto',
            '','')"/> </exchange>
```

fragmento de WS-CDL

- "Interactions": descripciones de intercambios, etiquetadas con un nombre de operación, equivalen a una operación WSDL
- Tiene lugar sobre un *canal*, tal como indica la declaración *channel variable*
- La relación entre participantes restringe la interacción a los roles que son válidos

fragmento de WS-CDL-II

```
1  <exchange name="RespuestaPresupuesto" informationType="tns:
       TipoRespuestaPresupuesto" action="respond">
2      <description type="documentation">
3          Intercambio Mensaje Peticion Presupuesto
4      </description>
5      <send variable="cdl:getVariable('peticionPrespuesto
           ','','')"/>
6      <receive variable="cdl:getVariable('peticionPrespuesto',
           '','')"/>
7  </exchange>
```

fragmento de WS-CDL – II

- “Exchanges": nombran el tipo de lo que va a ser intercambiado y la dirección del intercambio

fragmento de WS-CDL-III

```
1    <exchange name="FalloRespuestaPresupuesto" informationType="
         tns:TipoFalloRespuestaPresupuesto" action="respond"
         faultName="FalloProductoNoValido">
2        <description type="documentation">
3            Intercambio Mensaje Peticion Presupuesto
4    </description>
5        <send variable="cdl:getVariable('respuestaFallo','','')"
             />
6        <receive variable="cdl:getVariable('respuestaFallo
             ','','')"/>
7    </exchange>
8 </interaction>
```

Método de Descripción con WS-CDL

Definir los siguientes elementos:

1. Tipos–rol
2. Tipos–relación
3. Tipos–información
4. Tipos–token
5. Tipos-canal

Definición de roles

```
1  <roleType name="RolComprador">
2      <description type="documentation">
3          Rol del Comprador
4      </description>
5      <behavior name="ComportamientoComprador" interface="
           InterfazComportamientoComprador">
6          <description type="documentation">
7              Comportamiento del rol comprador
8          </description>
9      </behavior>
10 </roleType>
```

Definición de roles II

```
<roleType name="RolVendedor">
    <description type="documentation">
        Rol del Vendedor
    </description>
    <behavior name="ComportamientoVendedor" interface="
        InterfazComportamientoVendedor">
        <description type="documentation">
            Comportamiento del Vendedor
        </description>
    </behavior>
</roleType>
```

Definición de participantes I

```
<participantType name="Vendedor">
     <description type="documentation">
        Participante Vendedor
     </description>
     <roleType typeRef="tns:RolVendedor"/>
</participantType>
<participantType name="Comprador">
     <description type="documentation">
        Participante Comprador
     </description>
     <roleType typeRef="tns:RolComprador"/>
</participantType>
```

Definición de relaciones

```
<relationshipType name="CompradorParaVendedor">
    <description type="documentation">
        Relacion Comprador Vendedor
    </description>
    <roleType typeRef="tns:RolComprador"/>
    <roleType typeRef="tns:RolVendedor"/>
</relationshipType>
```

Definición de tokens

```
<token name="id" informationType="tns:TipoIdentidad">
    <description type="documentation">
        Token Identidad
    </description>
</token>
<token name="URI" informationType="tns:URI">
    <description type="documentation">
        Token Referencia para Canales
    </description>
</token>
```

Definición de tokens -II

```
1  <tokenLocator tokenName="tns:id" informationType="tns:
       TipoPeticionPresupuesto" query="/quote/@id">
2      <description type="documentation">
3          Identidad para Peticion de Presupuesto
4      </description>
5  </tokenLocator>
6  <tokenLocator tokenName="tns:id" informationType="tns:
       TipoRespuestaPresupuesto" query="/quote/@key">
7      <description type="documentation">
8          Identidad para Respuesta de Presupuesto
9      </description>
10 </tokenLocator>
11 <tokenLocator tokenName="tns:id" informationType="tns:
       TipoFalloRespuestaPresupuesto" query="/quote/@key">
12     <description type="documentation">
13         Identidad para Fallo Respuesta de Presupuesto
14     </description>
15 </tokenLocator>
```

Definición de canales

```
<channelType name="CompradorParaVendedorC">
    <description type="documentation">
        Tipo Canal Comprador para Vendedor
    </description>
    <roleType typeRef="tns:RolVendedor"/>
    <reference>
        <token name="tns:URI"/>
    </reference>
    <identity type="primary">
        <token name="tns:id"/>
    </identity>
</channelType>
```

WS-CDL Dinámico

La parte dinámica de WS-CDL sirve para definir el concepto de *coreografía* de SW, que es fundamentalmente un *caso de uso*

Coreografía:

- Conjunto de relaciones entre tipos rol, que actúan como una comprobación de tipos adicional sobre los canales
- Definición de conjuntos de variables, para instancias del canal y de los tipos–información
- Actividades

Coreografías

Forman una unidad de trabajo colaborativo

- Combinan acciones con características de *comportamiento común colaborativo* de los SW
  - Enumeran todas las interacciones de las relaciones binarias en las que actúan
  - Establecen la visibilidad de las variables–utilizando definiciones de variables
  - Prescriben patrones de comportamiento alternativo utilizando *unidades de trabajo/reacciones*
  - Hacen posible la recuperación desde fallos

Bloques constructivos de las coreografías

- Constructo *actividad*: describe el comportamiento de la coreografía. Es el bloque constructivo básico de los *"programas"* de WS-CDL
- *Excepción*: este bloque sirve para definir cómo se manejan las excepciones durante la ejecución de una *actividad*
- *Finalizador*: describe las acciones que hay que realizar cuando termina una actividad
- *Subcoreografías*

Coreografía – Sintaxis

```
choreography ::=
          <choreography  name="ncname"
                    complete="xsd:boolean␣XPath-expression"?
                    isolation="true"|"false"?
                    root="true"|"false"?
                    coordination="true"|"false"? >
              <relationship  type="qname" />+
          variableDefinitions?
              Activity
              <exception name="ncname">
                WorkUnit+
              </exception>?
              <finalizer name="ncname">
                        WorkUnit
              </finalizer>
           </choreography>
```

Conceptos uasados en coreografías de WS-CDL

- Variables: la información enviada o recibida durante una interacción se describe mediante una variable nombrada y un elemento *recordReference* opcional. Las variables contienen valores y tienen un tipo de información representado como un tipo
- Actividades: hay tres tipos de actividades, de flujo de control, unidades de trabajo y básicas. Las primeras constan de *secuencia*, *elección* y *actividades paralelas*. Una unidad de trabajo describe la ejecución condicional y repetida de una actividad. Una actividad basica incluye elementos como: Interacción, NoAcción, Acción silenciosa, Asignar y Realizar. La interacción es el elemento más importante de WS-CDL

Conceptos uasados en coreografías de WS-CDL-II

- Unidades de trabajo: describen las limitaciones que deben cumplirse para realizar las actividades. Tienen *guardas* y condiciones de repetición opcionalmente. definen actividades anidadas que se realizan cuando se evalúa que la condición de guarda es verdadera

Variables

- Cada variable se ubica en un *participante* y capturan información de los objetos de una colaboración
- Diferentes roles pueden compartir variables siempre que sean co-residentes en el mismo participante
- No pueden compartirse entre participantes diferentes

```
variableDefinitions ::=
          <variableDefinitions>
            <variable name="ncname"
                informationType="qname"?|channelType="qname"
                mutable="true|false"?
                free="true|false"?
                silent-action="true|false"?
                roleTypes="listOfQname"? />+
          </variableDefinitions>
```

Variables – II

- Influyen en el progreso de las colaboraciones entre participantes
- Tipos de variables:
  - Variables de intercambio de información: definir instancias de los documentos intercambiados entre los roles
  - Variables de estado: definir instancias de la información de estado en un rol
  - Variables de canal: definir instancias de los tipos–canal
- Definiciones de variables:
  - Especificar el tipo de valor que contiene una variable utilizando *informationType*, *channelType*
  - Especificar el rol de un participante (donde reside la variable) que colabora con otros

Variables – III

Valor de una variable

- se inicializa antes de comenzar una coreografía y está disponible para todos los roles
- las variables compartidas (sólo dentro de 1 participante) contienen información común a 2 ó más roles
- se pueden hacer disponibles en un rol asignándolas como resultado de una interacción
- pueden hacerse disponibles en un rol generándolas con una expresión *XPath*
- variables definidas localmente contienen información creada y cambiada por un rol; pueden ser *intercambio de información*, *estado* y *variables de canal*
- se pueden utilizar para determinar las decisiones y acciones que se han de tomar dentro de una coreografía

Actividades

Propósito

Permiten la comunicación y el *alineamiento de información* entre participantes que colaboran

Tipos de actividades en WS-CDL

1. Actividades básicas
2. Unidades de trabajo
3. Actividades estructurales

Traducción de actividades básicas de WS-CDL a un modelo de eventos

WS-CDL basic activity	**Event-B concepts**
<assign $roleType$ = "$qname$" > <copy $name$ = "$ncname$" > <source $variable$ = "var_name1" / > <target $variable$ = "var_name2" / > </copy> < /assign>	WHERE Any THEN $var\_name2 := var\_name1$ END
<exchange $name$ = "$exchange\_name$" $informationType = infoType$ $action$ = "$request$" > <send $variable = var1$/ > < $receivevariable = var2$/ > </exchange>	INVARIANTS $var1 \in infoType$ and $var2 \in infoType$ EVENT WHEN $dom(msg) = \{i\}$ THEN $msg = \{i + 1 \mapsto exchange\_name\}$ END

Actividades - sintaxis (incompleta)

```
Activity-Notation ::=
  <sequence>
    <interaction  name="ncname"
          channelVariable="qname"
          operation="ncname">
          <description;></description>
          <participate  relationshipType="qname"
             fromRoleTypeRef="qname" toRoleTypeRef="qname" />
          <exchange  name="ncname" faultName="qname"?
             informationType="qname"?|channelType="qname"?
             action="request"|"respond" >
             <send variable="XPath-expression"?/>
             <receive variable="XPath-expression"?/>
          </exchange>*
       </interaction>*
  </sequence>
```

Actividades Básicas

Clases de actividades de este tipo

- Interacciones: comunicación binaria entre 2 tipos–rol
- Asignación de variables
- Sentencia “perform”: inicio de una subcoreografía

Actividades Básicas: Interacción

Intercambio de mensajes entre 2 roles en una relación

- petición y aceptación de una operación a través de un canal común
  - de una vía u operación *request-response*
  - flujo de información (dirección de petición “fromRole”: “toRole”; dirección de respuesta “toRole” : “fromRole”)

Registro del estado en un rol

- Crear nuevas variables y modificarlas dentro de un rol

Actividades Básicas: Interacción II

Ejemplo de Interacción

```
<interaction name="crearPO"
    channelVariable="tns:CanalMinorista"
    operation="gestionarOrdenCompra">
    <participate relationshipType="tns:
        RelacionConsumidorMinorista"
fromRoleTypeRef="tns:Consumidor" toRoleTypeRef="tns:Minorista"/>
     <exchange name="request"
        informationType="tns:tipoOrdenCompra" action="request">
        <send variable="cdl:getVariable('tns:ordenCompra','␣','␣
            ')"/>
        <receive variable="cdl:getVariable('tns:ordenCompra','␣
            ','␣')"
          recordReference="informacionregistrodelcanal"/>
      </exchange>  </interaction>
```

Actividades Básicas: Interacción – III

Alineamiento de información

- se trata de poner de acuerdo a los roles de una interacción en cosas comunes, tales como:
  - cambios de estado de las variables que residen en un rol respecto de los cambios de estado de las variables de otro
  - valores de los mensajes intercambiados desde un rol a otro

Actividades Básicas: Interacción – Sintaxis

```
<interaction name="ncname" channelVariable="qname"
  operation="ncname"  time-to-complete="xsd:duration"?
          align="true"|"false"?
          initiateChoreography="true"|"false"?>
 <participate relationship="qname"
          fromRole="qname"toRole="qname"/>
 <exchange mesaggeContentType="qname"action="request"|"respond">
        <use variable="XPath-expression"/>
        <populate variable="XPath-expression"/>
 </exchange>*
 <record name="ncname" role="qname" action="request"|"respond">
        <source variable="XPath-expression"/>
        <target variable="XPath-expression"/>
 </record>*
</interaction>
```

Fragmento de coreografía

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <package
3      name="Ejemplo Inicial"
4      author="M.I.C."
5      version="1.0"
6      targetNamespace="http://www.w3.org/2002/ws/chor/primer"
7      xmlns:tns="http://www.w3.org/2002/ws/chor/primer"
8      xmlns="http://www.w3.org/2005/10/cdl"
9      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10     xmlns:primer="http://www.w3.org/2002/ws/chor/primer">
11     <description type="documentation">
12         Ejemplo inicial para comenzar
13     </description>
14     ...
```

Fragmento de coreografía -II

```
1  <choreography name="CoreografiaEjemploInicial" root="true">
2   <description type="documentation">
3    La coreografia para el caso de uso del ejemplo inicial
4   </description>
5   <sequence>
6      <interaction name="ObtencionPresupuesto"
7        operation="obtenerPresupuesto"
8        channelVariable="tns:CompradorParaVendedorC">
9        <description type="documentation">
10         Obtencion de Presupuesto
11       </description>
12       <participate relationshipType="tns:CompradorParaVendedor"
13         fromRoleTypeRef="tns:RolComprador"
14         toRoleTypeRef="tns:RolVendedor"/>
```

Fragmento de coreografía -III

```
1  <exchange name="PeticionPresupuesto"
2    informationType="tns:TipoPeticionPresupuesto" action="
         request">
3    <description type="documentation">
4        Intercambio Mensaje Peticion Presupuesto
5    </description>
6    <send variable="cdl:getVariable('peticionPresupuesto
         ','','')"/>
7    <receive variable="cdl:getVariable('peticionPresupuesto
         ','','')"/>
8  </exchange>
```

Fragmento de coreografía -IV

```
1  <exchange name="RespuestaPeticionPresupuesto"
2    informationType="tns:TipoRespuestaPeticionPresupuesto"action
         ="respond">
3    <description type="documentation">
4      Intercambio Mensaje Respuesta Peticion de Presupuesto
5    </description>
6    <send variable="cdl:getVariable('respuestaPresupuesto
         ','','')"/>
7    <receive variable="cdl:getVariable('respuestaPresupuesto
         ','','')"/>
8   </exchange>
```

Fragmento de coreografía-V

```
1     <exchange name="FalloRespuestaPeticionPresupuesto"
2      informationType="tns:TipoRespuestaPeticionPresupuesto"
3      action="respond" faultName="FalloProductoNoValido">
4      <description type="documentation">
5         Intercambio Fallo Respuesta Peticion Presupuesto
6      </description>
7      <send variable="cdl:getVariable('respuestaFallo','','')"/>
8      <receive variable="cdl:getVariable('respuestaFallo','','')"
           />
9     </exchange>
10   </interaction>
11  </sequence>
12 </choreography>
13 </package>
```

Actividades Básicas: Asignación

Registro del estado en un rol

- Crear o modificar variables y hacerlas disponibles dentro de un tipo rol utilizando el valor de otra variable o expresión

```
<assign roleType="qname">
  <copy name="ncname" causeException="qname"?>
   <source variable="XPath-expression"|expression="XPath-
       expression"?/>
   <target variable="XPath-expression"/>
  </copy>+
</assign>
```

Asignación:reglas semánticas

- Compatibilidad entre `source` y `target`
- Ubicación de ambos cuando el target se define como `variable`
- Sólo utilización de `getVariable()` si el atributo es `silent`
- Orden de ejecución de los elementos `copy`
- Semántica de completación de la sentencia `assign`
- Como máximo 1 elemento con el atributo optativo `causeException`

Assign– ejemplos

```
<assign roleType="tns:Minorista">
  <copy name="copiarInfoDireccion">
    <source variable="cdl:getVariable('MsjOrdenCompra','','/PO,
        DireccionCliente')" />
    <target variable="cdl:getVariable('DireccionCliente','','')"
        />
  </copy>
</assign>
```

```
<assign roleType="tns:Minorista">
  <copy name="copiarInfoPrecio">
    <source expression="(10+237)/34" />
    <target variable="cdl:getVariable('PrecioProducto','','','
        tns:Minorista')" />
  </copy>
</assign>
```

Acciones silenciosas

- `silentAction` indica el punto en el que las acciones de un participante no son observadas por los otros
- El atributo optativo `roleType` que indica dónde ha de realizarse ha de ser especificado, si no se convertirán en silenciosas las acciones referidas en todos los roles de la coreografía actual

```
1    <silentAction roleType="qname"?/>
```

noAction

- Similar a `silentAction`, se utiliza cuando se necesita *sintácticamente* una acción, pero no se le requiere que realice ninguna actividad

```
1  <noAction roleType="qname?␣/>
```

noAction –ejemplo

```
<exceptionBlock name="manejarExcepcionTimeout">
      <workunit  name="manejarExcepcionTimeout" >
                 <noAction>
      </workunit>
</exceptionBlock>
```

Unidades de trabajo

Idea general sobre el constructo

- Realización de un modelo de computación “dirigido por eventos”
- Una *regla de reacción*, que ha de satisfacerse para obtener progreso normal/causar excepción, se encarga de *guardar* un conjunto de potenciales actividades, prescribiendo las restricciones sobre la información que sean necesaria

Unidades de trabajo II

- Cada constructo *workunit* de WS-CDL:
  1. una *guarda*,
  2. un bloque iterativo que encierra la actividad
- La actividad se activa para ejecución dependiendo del valor de certeza de la guarda
- Bloqueo de la guarda hasta que la condición sea cierta (estrategia *bloqueante*)
- Tras la ejecución de la actividad: evaluación de las condiciones guarda y repetición del bucle

Traducción de unidades de trabajo de WS-CDL a un modelo de eventos

WS-CDL work-unit	**Event-B Event**
<workunit $name =$ "$unitname$"	EVENT $unitname$
$guard =$ "$xsd : booleanXPath - expression$"?	WHEN $guard$ and $repeat$
$repeat =$ "$xsd : booleanXPath - expression$"?	THEN
Activity-Notation	body
< /workunit>	END

Unidades de trabajo III

Guardas de reacción

- expresan el interés en la disponibilidad de 1 ó más informaciones que afectan a las variables
- Cuando la variable se convierte en disponible y la condición de la guarda es cierta, las actividades dependientes son activadas
- Una condición guarda puede depender de variables ubicadas en un un rol diferente del que la contiene

Unidades de trabajo IV

`repeat`

Marca la reactivación de una unidad de trabajo

`isAligned(var1,var2,rel)`

Especifica la necesidad de alineamiento de 2 roles dentro de una relación

Unidades de trabajo– Sintaxis

```
<workunit name="ncname"
      guard="xsd:boolean XPath-expression"?
      repeat="xsd:boolean XPath-expression"?
      block="true|false">

      Activity
</workunit>
```

Unidades de trabajo – Ejemplo

```
<workunit name="unit" block="false"
    guard="not(cdl:isVariableAvailable('tns:ordenCompra','tns:
        Consumidor'))">
    <assign roleType="tns:Consumidor">
      <copy name="copia1">
       <source expression="cdl:doc('&lt;PO&gt; &lt;idOrden name
           =&quot;10&quot;/&gt;&lt;RefCliente name=&quot;1000&
           quot;/&g20"
       <target variable="cdl:getVariable( 'tns:ordenCompra','
           ',' ')"/>
      </copy>
    </assign>
</workunit>
```

Actividades estructurales

Clases de actividades

- *Secuencia*: especifica que sus argumentos sean evaluados secuencialmente
- *Paralela*: permite ejecución en paralelo de sus argumentos
- *Elección*: parecida a la *selección-no determinista*, selecciona una de las *ramas activas* (guardas) para ejecutar
- WS-CDL vs. WS-BPEL: el primero tiene operadores parecidos a los de π–Calculus; el segundo sigue un modelo de flujos concurrentes de trabajos

Ejemplo: Descripción de un escenario de órdenes de compra

Enunciado:

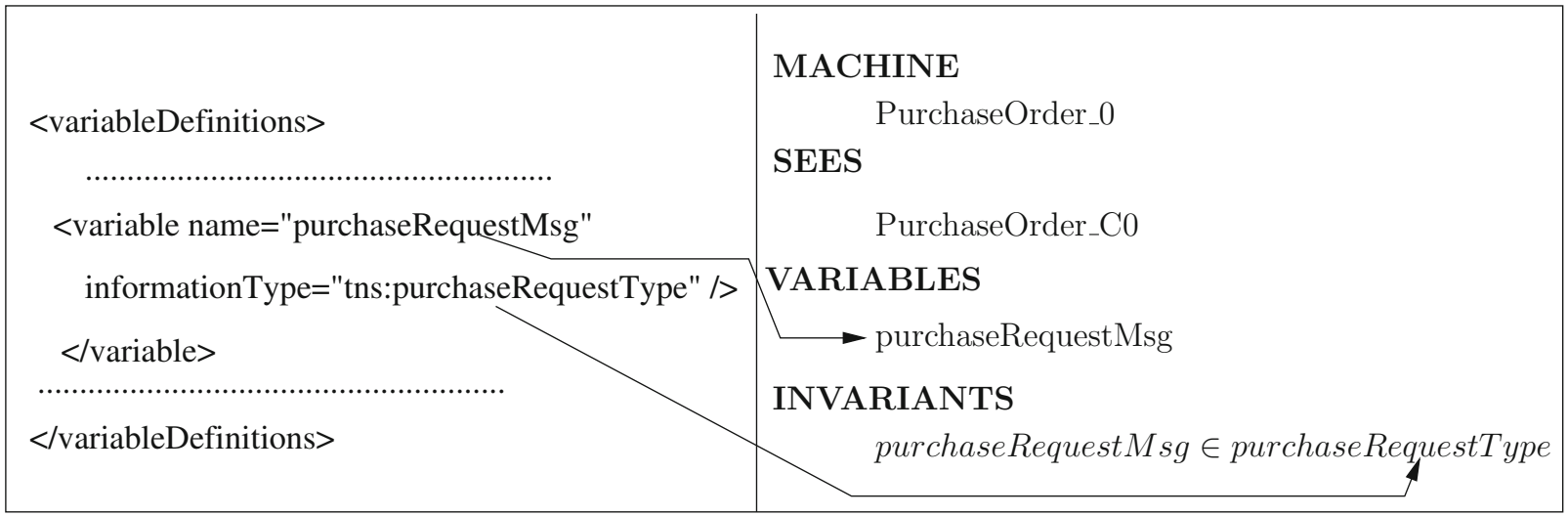
Este escenario involucra a cuatro participantes: comprador, vendedor, tarjeta de crédito y almacén. El Comprador inicia la coreografía enviando una solicitud de compra a el Vendedor con la información del producto y la tarjeta de crédito. El vendedor solicita después comprobar si exister el producto en el almacén y la validez de la información de la tarjeta de crédito. Si ambos resultados son positivos, el vendedor responderá con una confirmación de la orden de compra; en caso contraio, el Vendedor rechaza la solicitud.

Transformación de la parte estática de la coreografía WS-CDL a un modelo de eventos

```
<informationType name="purchaseRequestType"
   type="tns:purchaseRequestMsg"/>
 .....
<roleType name="BuyerRole">
<roleType name="CreditCardRole">
<roleType name="SellerRole">
<roleType name="StoreRole">
.....
<relationshipType name="Buyer_Seller">
   <role type="BuyerRole" behavior="serviceRequester"/>
   <role type="SellerRole" behavior="serviceProvider"/>
</relationshipType>
```

CONTEXT
PurchaseOrder_C0
SETS
Services
purchaseRequestType
CONSTANTS
Requesters
Providers
BuyerRole
SellerRole
CreditCardRole
StoreRole
AXIOMS
$Requesters = \{BuyerRole, SellerRole\}$
$Providers = \{SellerRole, CreditCardRole, StoreRole\}$
$Relationships = \{BuyerRole \mapsto SellerRole, ..\}$

Transformación de las variables WS-CDL a un modelo de eventos



Transformación de las actividades y unidades de trabajo WS-CDL a un modelo de eventos

```
<interaction channelVariable="Buyer2SellerC"
   name="purchaseRequest" operation="purchaseOrder">
  <participate toRole="SellerRole"
      relationshipType="BuyerSeller" fromRole="BuyerRole"/>
   <exchange action="request" name="purchaseRequest"
       informationType="purchaseRequestMsg">
     <send variable="cdl:getVariable("purchaseReq","","")""/>
     <receive variable="cdl:getVariable("purchaseReq","","")"/>
   </exchange>

 .....

<workunit name="Send purchase confirmation" repeat="false"
    guard="creditCardValidity>0 and storeAmount>0 ">
  <exchange action="respond" name="purchaseConfirm">
      .....
  </exchange>
```

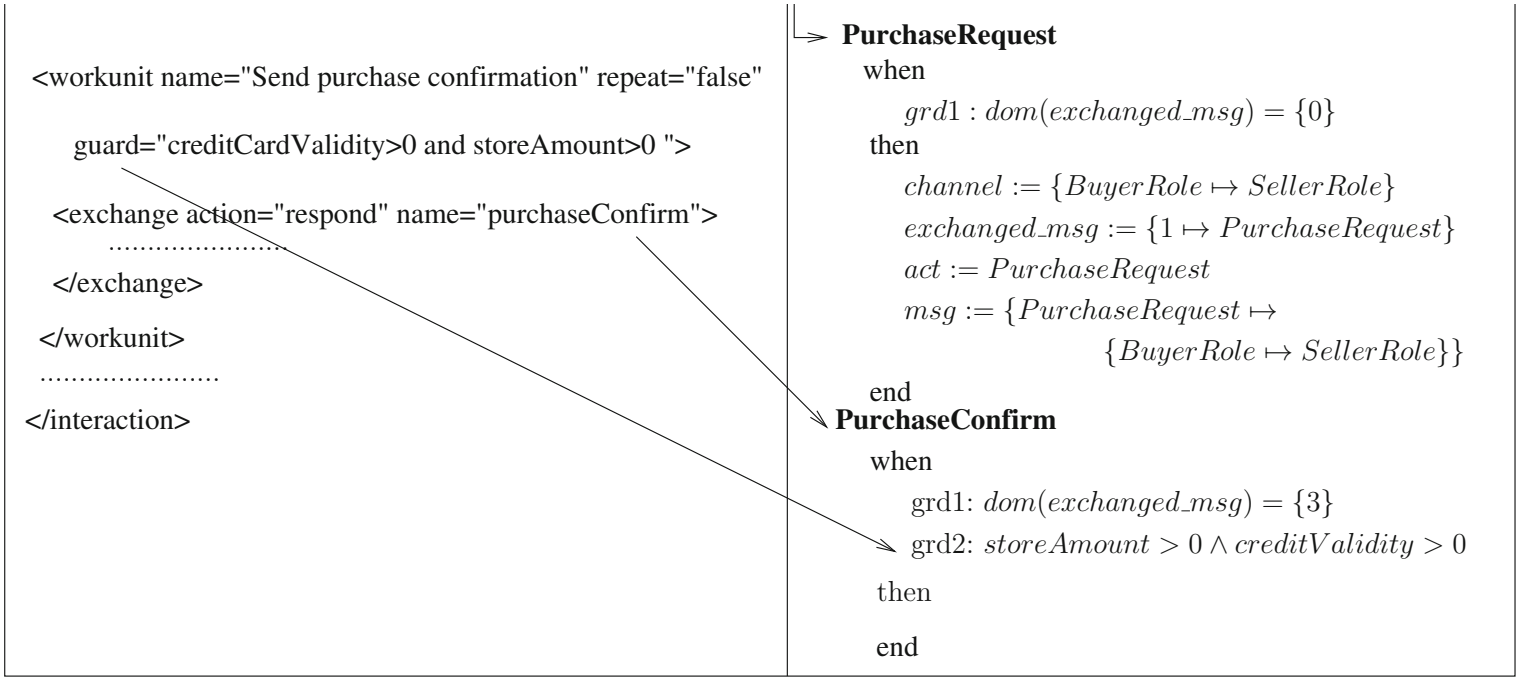
VARIABLES
exchanged_msg
channel
msg
act
purchaseReq

INVARIANTS
$purchaseReq \in purchaseRequestMsg$
$exchanged\_msg \in OrderedMessages$
$msg \in Actions \longrightarrow Relationships$
$card(channel) = 1$
$channel \in Relationships$
dom(msg)=ran(exchanged_msg)
$msg = \{act \mapsto channel\}$
$card(msg) = 1$

EVENTS

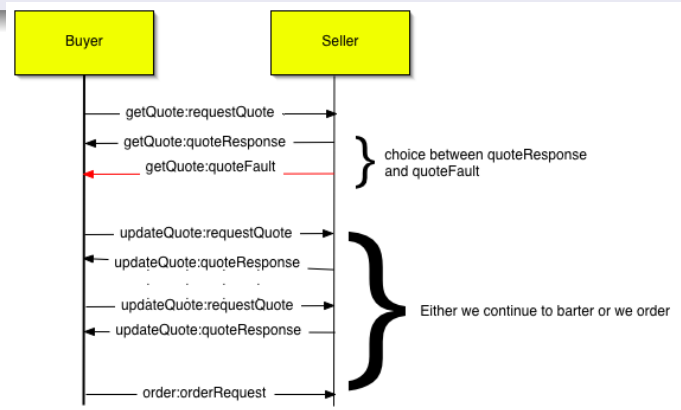
PurchaseRequest
when
$grd1 : dom(exchanged\_msg) = \{0\}$
then
$channel := \{BuyerRole \mapsto SellerRole\}$
$exchanged\_msg := \{1 \mapsto PurchaseRequest\}$
$act := PurchaseRequest$

Transformación de las actividades y unidades de trabajo WS-CDL a un modelo de eventos-II



Ejercicio Propuesto

Hemos ampliado la lógica de negocio del ejemplo inicial (WS-CDL) de acuerdo con el diagrama de secuencia UML:



Ahora el comprador pide un precio al vendedor y entra en algún tipo de comportamiento repetitivo en el cuál pide un precio actualizado al vendedor y continúa de esta manera hasta que decide comprar el artículo al mejor precio. Se pide desarrollar la descripción completa de la coreografía utilizando WS-CDL.

Composición de coreografías

Idea fundamental

- WS-CDL posee un modelo *composicional* que permite obtener un modelado escalable del comportamiento colaborativo de los SW

Composición de coreografías II

Características de la composicionalidad de WS-CDL

1. Inicialmente, construir coreografías pequeñas
2. Combinarlas juntas para formar una coreografías mayor
3. Después, realizar proyecciones sobre los participantes para conseguir obtener el punto de vista de estos:
   - La composición hace que las proyecciones sobre los participantes se puedan obtener más fácilmente
   - Sin tener la visión de conjunto de un “proceso de negocio” las proyecciones de los participantes no tienen ningún sentido

Composición de coreografías: sentencia `perform`

Recursividad

Las coreografías se pueden combinar recursivamente para formar nuevas coreografías

```
<perform   choreographyName="qname"
           choreographyInstanceId="XPath-expression"?
           block="true|false"? >
   <bind   name="ncname">
     <this variable="XPath-expression" roleType="qname"/>
     <free variable="XPath-expression" roleType="qname"/>
   </bind>*
   Choreography-Notation?
</perform>
```

Composición de coreografías: sentencia `perform`–II

Definición

La actividad `perform` hace posible especificar dentro de una coreografía que otra va a actuar en ese punto de su definición, como una coreografía *contenida*

Elementos de la definición sintáctica

- `bind`: permite compartir información con la coreografía *contenida*
- `roleType` crean alias de la coreografía contenida hacia la *contenedora*
- `this` indica la variable de la contenedora que se liga al elemento `free` de la *contenida*

Ejemplo de sentencia `perform`

```
<choreography name="CoreografiaMinoristaAlmacen">
    <variableDefinitions>
       <variable name="ordenCompra"
   informationType="ordenCompra" roleTypes="tns:Minorista" free=
       "true"/>
    </variableDefinitions>
      ...
</choreography>
```

Ejemplo de sentencia `perform`– II

```
1  <choreography name="CoreografiaCompra">
2     <variableDefinitions>
3        <variable name="ordenCompraMinorista"
4                  informationType="ordenCompra" roleTypes="tns:
                       Minorista"/>
5     </variableDefinitions>
6     <perform choreographyName="CoreografiaMinoristaAlmacen">
7        <bind name="aliasMinorista">
8          <this variable= "cdl:getVariable('tns:
                ordenCompraMinorista','','')"
9                roleType="tns:Minorista"/>
10         <free variable="cdl:getVariable('tns:ordenCompra','','')
                " roleType="tns:Minorista"/>
11       </bind>
12    </perform>
13 </choreography>
```

Finalización de una coreografía

- Utilización en coreografías *contenidas*
- Sólo si la coreografía contenedora define `performs` con bloques finalizadores podría definir actividades de este tipo
- Al menos 1 de los bloques finalizadores ha de terminar para cada coreografía *contenida*
- Una actividad de finalización realizada sobre bloques de finalización no instalados, no tendrá efecto

Finalización de una coreografía – II

```
<finalize   name="ncname"? choreographyName="ncname"
              choreographyInstanceId="XPath-expression"?
              finalizerName="ncname"? />
```

- La actividad finalizadora activa un bloque finalizador en una instancia de la coreografía *contenida* de nivel más cercano
- Es obligatorio indicar `finalizerName` si la coreografía contenida a la que aplica posee más de un bloque finalizador definido

Finalización de una coreografía-Ejemplo

```
1  <choreography name="CreditDecider">
2       <parallel>
3          <perform name="creditForA"
4               choreographyName="CoordinatedCreditAuthorization"
5               choreographyInstanceId="'creditForA'">
6               <!-- bind such that this does the business for A -->
7          </perform>
8          <perform name="creditForB"
9               choreographyName="CoordinatedCreditAuthorization"
10              choreographyInstanceId="'creditForB'">
11              <!-- bind such that this does the business for B -->
12         </perform>
13        </parallel>
```

Finalización de una coreografía-Ejemplo – II

```
<workunit name="chooseA"
        guard="cdl:getVariable('Chosen','','','Broker')='A'" >
        <finalize choreographyName="
            CoordinatedCreditAuthorization"
                choreographyInstanceId="'creditForA'"
                finalizerName="drawDown"/>
        <finalize choreographyName="
            CoordinatedCreditAuthorization"
                choreographyInstanceId="'creditForB'"
                finalizerName="replenish"/>
    </workunit>
```

Finalización de una coreografía-Ejemplo – III

```
1   <workunit name="chooseB"
2       guard="cdl:getVariable('Chosen','','','Broker')='B'" >
3       <finalize choreographyName="
            CoordinatedCreditAuthorization"
4               choreographyInstanceId="'creditForB'"
5               finalizerName="drawDown"/>
6       <finalize choreographyName="
            CoordinatedCreditAuthorization"
7               choreographyInstanceId="'creditForA'"
8               finalizerName="replenish"/>
9   </workunit>
```

Finalización de una coreografía-Ejemplo – IV

```
<workunit name="chooseNeither"
        guard="cdl:getVariable('Chosen','','','Broker')='0'" >
        <finalize choreographyName="
            CoordinatedCreditAuthorization"
                choreographyInstanceId="'creditForA'"
                finalizerName="replenish"/>
        <finalize choreographyName="
            CoordinatedCreditAuthorization"
                choreographyInstanceId="'creditForB'"
                finalizerName="replenish"/>
    </workunit>


</choreography>
```

Package, Import, Templates

Se trata de facilidades para conseguir modularidad y reutilización de coreografías

Package

Características

- Permite combinar varios tipos de coreografías en un mismo espacio de nombres:
  - `informationType`, `token`, `tokenLocator`
  - `role`, `relationship`, `participant`
  - `channelType`
  - definiciones de variables
  - coreografía

Package – sintaxis

```
<package>
    name="ncname"
    author="xsd:string"
    version="xsd:string"
    targetNamespace="URI"
    xmlns="http://www.w3.org/ws/choreography/2004/02/WSCDL/"/>
    importDefinitions*
    token*
    tokenLocator*
    role*
    relationship*
    participant*
    chanelType*
    Choreography*
</package>
```

Import

Características

- Una importación permite utilizar tipos-coreografía que están definidos en otro paquete coreográfico

```
<importDefinitions>
    <import namespace="URI" location="URI"/>+
</importDefinitions>
```

Templates

Idea fundamental

La utilización de *templates* permite especificar grados variables de detalle incrementalmente, permitiendo, por tanto, conseguir la reutilización de coreografías en distintos contextos: industria, local, etc.

Ejercicio Propuesto:Solución

Ahora el comprador pide un precio al vendedor y entra en algún tipo de comportamiento repetitivo en el cuál pide un precio actualizado al vendedor y continúa de esta manera hasta que decide comprar el artículo al mejor precio.

```
1  Boolean regateoHecho@Vendedor = false
2     Elicitar una oferta del vendedor
3        repetir
4        {
5           elegir
6           {
7              { Aceptar la oferta y confirmar pedido
8                regateoHecho@Vendedor = true
9              }
10             { Rechazar la oferta y pedir una nueva
11             }
12          }
13       } hasta (regateoHecho@Vendedor = true)
```

Tipos de información definidos

```
1  <informationType name="QuoteRequestType" type="primer:
       QuoteRequestMsg">
2  <description type="documentation">Quote Request Message</
       description></informationType>
3  <informationType name="QuoteResponseType" type="primer:
       QuoteResponseMsg">
4  <description type="documentation">Quote Response Message</
       description></informationType>
5  <informationType name="URI" type="xsd:uri">
6  <description type="documentation">Reference Token For Channels</
       description></informationType>
7  <informationType name="OrderRequestType" type="primer:
       OrderRequestMessage">
8  <description type="documentation">Order Request Message</
       description></informationType>
9  <informationType name="OrderResponseType" type="primer:
       OrderResponseMessage">
10 <description type="documentation">Order Response Message</
```

Tokens y Token Locators

```
1  <token name="id" informationType="tns:IdentityType">
2  <description type="documentation">Identity token</description></
       token>
3  <token name="URI" informationType="tns:URI">
4  <description type="documentation">Reference Token for Channels</
       description></token>
5  <tokenLocator tokenName="tns:id" informationType="tns:
       QuoteRequestType" query="/quote/@id">
6  <description type="documentation">Identity for QuoteRequestType
       </description></tokenLocator>
7  <tokenLocator tokenName="tns:id" informationType="tns:
       QuoteResponseType" query="/quote/@key">
8  <description type="documentation">Identity for QuoteResponseType
       </description></tokenLocator>
9  <tokenLocator tokenName="tns:id" informationType="tns:
       QuoteResponseFaultType" query="/quote/@key">
10 <description type="documentation">Identity for
       QuoteResponseFaultType</description></tokenLocator>
```

Tipos rol

```
<roleType name="BuyerRole">
<description type="documentation">Role for Buyer</description>
<behavior name="BuyerBehavior"interface="BuyerBehaviorInterface"
    >
<description type="documentation">Behavior for Buyer Role
</description></behavior>
</roleType>
<roleType name="SellerRole">
<description type="documentation">Role for Seller</description>
<behavior name="SellerBehavior" interface="
    SellerBehaviorInterface">
<description type="documentation">Behavior for Seller
</description></behavior>
</roleType>
```

Tipos relación

```
<relationshipType name="Buyer2Seller">
 <description type="documentation">Buyer Seller Relationship</
     description>
 <roleType typeRef="tns:BuyerRole"/>
 <roleType typeRef="tns:SellerRole"/>
</relationshipType>
```

Tipos participante

```
<participantType name="Seller">
  <description type="documentation">Seller Participant</
      description>
  <roleType typeRef="tns:SellerRole"/>
</participantType>
<participantType name="Buyer">
  <description type="documentation">Buyer Participant</
      description>
  <roleType typeRef="tns:BuyerRole"/>
</participantType>
```

Tipos canal

```
<channelType name="Buyer2SellerChannel">
<description type="documentation">Buyer to Seller Channel Type</
    description>
<roleType typeRef="tns:SellerRole"/>
<reference>
  <token name="tns:URI"/>
</reference>
<identity type="primary">
  <token name="tns:id"/>
</identity>
</channelType>
```

Definición de la coreografía con WS-CDL

```
1 <choreography name="IntermediateChoreography" root="true">
2  <description type="documentation">...</description>
3  <relationship type="tns:Buyer2Seller"/>
4  <variableDefinitions>
5   ...
6     <variable name="regateoHecho" informationType="tns:Boolean"
          roleTypes="tns:SellerRole">
7     <description type="documentation">...</description>
8     </variable>
9  </variableDefinitions>
```

Elicitar oferta del vendedor

```
<sequence>
  <assign roleType="tns:SellerRole">
  <description type="documentation">Initialise Loop Variable</
      description>
  <copy name="setBarteringDone">
     <description type="documentation">Set barteringDone to
         false</description>
     <source expression="false()"/>
     <target variable="cdl:getVariable('barteringDone','','')"/>
  </copy>
  </assign>
  <interaction name="QuoteElicitation" operation="getQuote"
     channelVariable="tns:Buyer2SellerC">
     <description type="documentation">Elicit a quote from the
         seller</description>
     <participate relationshipType="tns:Buyer2Seller"
         fromRoleTypeRef="tns:BuyerRole"  toRoleTypeRef="tns:
         SellerRole"/>
```

El

```
...
<interaction name="QuoteElicitation" operation="getQuote"
    channelVariable="tns:Buyer2SellerC">
 <description type="documentation">Elicit a quote from the
     seller</description>
 <participate relationshipType="tns:Buyer2Seller"
     fromRoleTypeRef="tns:BuyerRole" toRoleTypeRef="tns:
     SellerRole"/>
 <exchange name="QuoteRequest" informationType="tns:
     QuoteRequestType" action="request">...</exchange>
 <exchange name="QuoteResponse" informationType="tns:
     QuoteResponseType" action="respond">...          </
     exchange>
<exchange name="QuoteResponseFault" informationType="tns:
    QuoteResponseFaultType" action="respond" faultName="
    InvalidProductFault">
  <send variable="cdl:getVariable('faultResponse','','')"
      causeException="TerminalFailure"/>
  <receive variable="cdl:getVariable('faultResponse','','')"
      causeException="TerminalFailure"/>
</exchange>
</interaction>
...
```

```
<workunit name="WhileBarteringIsNotFinished"
    guard="cdl:getVariable("barteringDone","","")= true()" repeat
        ="true()">
3   <description type="documentation">While barteringDone is
        false</description>
4   <choice>
5      <sequence>
6      <description type="documentation">...</description>
7      <interaction name="QuoteAccept" operation="order"
           channelVariable="tns:Buyer2SellerC">
8      <description type="documentation">...</description>
9      <participate relationshipType="tns:Buyer2Seller"
           fromRoleTypeRef="tns:BuyerRole" toRoleTypeRef="tns:
           SellerRole"/>
10      <exchange name="OrderRequest" informationType="tns:
            OrderRequestType" action="request">
11          <send variable="cdl:getVariable('orderRequest','','')
                "/>
12          <receive variable="cdl:getVariable('orderRequest
                ','','')"/>
13      </exchange>
14      </interaction>
```

Unidad de trabajo "WhileBarteringIsNotFinished-II

```
1  ...
2          <assign roleType="tns:SellerRole">
3             <description type="documentation">Break out of the
                  loop</description>
4             <copy name="setBarteringDone">
5             <description type="documentation">Set barteringDone to
                   true</description>
6                <source expression="true()"/>
7                <target variable="cdl:getVariable('barteringDone
                      ','','')"/>
8             </copy>
9          </assign>
10  </sequence>
```

Unidad de trabajo "WhileBarteringIsNotFinished-III

```
<sequence>
  <description type="documentation">Reject the quote and ask for
       a new quote</description>
  <interaction name="QuoteReelicitation" operation="updateQuote"
       channelVariable="tns:Buyer2SellerC">
  <description type="documentation">Barter based on previous
      quote</description>
  <participate relationshipType="tns:Buyer2Seller"
      fromRoleTypeRef="tns:BuyerRole" toRoleTypeRef="tns:
      SellerRole"/>
  <exchange name="QuoteRequest" informationType="tns:
      QuoteRequestType" action="request">
  <description type="documentation">Quote re-request based on
      amended quoteRequest</description>
     <send variable="cdl:getVariable('quoteRequest','','')"/>
     <receive variable="cdl:getVariable('quoteRequest','','')"/>
  </exchange>
  ...
```

Unidad de trabajo "WhileBarteringIsNotFinished-IV

```
...
      <exchange name="QuoteResponse" informationType="tns:
          QuoteResponseType" action="respond">
       <send variable="cdl:getVariable('quoteResponse','','')"/>
       <receive variable="cdl:getVariable('quoteResponse','','')"
           />
      </exchange>
      </interaction>
     </sequence>
    </choice>
   </workunit>

  </sequence>
</choreography>
```