WS-CDL

Web Services Choreography Description Language

ReDCAD Laboratory, University of Sfax, Tunisia

1 Overview

The choreography has been introduced as a new view on the services interaction. It's a description of abstract protocols [6]. It offers a collaborative decentralized coordination. Choreography, which is descriptive in nature, describes the interaction contract between two or more web services. It helps to describe the services behavior in the composition.
Choreography languages reflect the long-term interactions. They allow the user to describe the peer-to-peer collaboration between services by defining their common observable behavior. In the realm of choreography, two different modeling approaches differ:

Interconnection models based approach The control flow is defined by participant. The behavior of each participant and the exchanged messages are represented. However, the models could be incompatible.

- BPMN (Business Process Modeling Notation) [7] it's a platform independent language. It uses a typical notation for every process. To make interconnecting different processes, this language uses messages flows. In that way, all interactions are listed with the definition of the control flow between them. Hence, communications are established using data objects and communicating activities. Nevertheless, BPMN doesn't support multiple instances of participant. So, to solve this problem, it uses a Pool set to represent multiple participants in one conversation and uses PBD (Participant Behavior Description) as views from the individual partners.

- BPEL4Chor [4] It's a BPEL extension and based on "Abstract Process Profile for observable Behavior". BPEL4Chor uses abstract processes and supports all the different choreographies design phases. It ensures communication between participants using message links.

Interaction models based approach The elementary interactions, namely demand and the exchange of request-response messages, are the basic elements of this approach. Behavioral dependencies, shown in these interactions and combinations of interactions, are grouped in complex interactions. According to this type, we can list WS-CDL and Let's Dance:

- Let's Dance [8] Its specification is based on graph. It's also based on independent visual notation. The communication action is performed by an actor playing a role. The exchanged messages are message sending and message receipt action.
- WS-CDL: its specification is based on an XML document [3]. It defines a common behavior for all participants and models a message between two (or more) participants. It provides more detailed models.

Each of these languages has brought new concepts, and redefined ones already known by others. This created a multitude of concepts, sometimes overlapping and a multitude of ways to manipulate. Unlike BPEL, BPEL4Chor, WS-CDL provides a description of the collaboration between the partners involved independently in the process.

2 WS-CDL constructors

WS-CDL is used to describe the common and collaborative observable behavior of multiple services which interact to achieve a goal [1]. It is a language relying on layers with different levels of expressibility to describe a choreography. However, it is not necessary to use all the features of WS-CDL to provide correct choreography specification [2]. WS-CDL is composed of two parts[5] one describing static relationships which are invariant to characterize data types, types of communication channels, types of communication partners, etc...The second part describes the dynamic behavior: interactions between communication partners (web service) and their temporal dependencies.

2.1 Static Part

The static part defines the entities that collaborate in a WS-CDL specification. They are defined as follows:

2.1.1 RoleType

<RoleType> defines the observable behavior of a participant services. It specifies the implementation process given by a participant with a specific role.

Listing 1: RoleType General Structure

```
< roleType name="ncname" >
  <description type=" documentation" </description> ?
  <behavior name="ncname" interface="qname" ? /> +
</roleType>
```

2.1.2 ParticipantType

<ParticipantType> represents an entity playing a particular set of roles in the choreography and logically regroups observable behaviors of roles

Listing 2: ParticipantType General Structure

```
<participantType name="NCName">
  <roleType typeRef="QName" /> +
</participantType>
```

2.1.3 RelationshipType

Once we have defined roles, we can define relationships. In WS-CDL, a relationship declares its intention to interact between two roles. So <relationshipType> identifies RoleType and behaviors.

Listing 3: RelationshipType General Structure

```
< relationshipType name="ncname">
  <role type="qname" behavior="list of ncname" ? />
  <role type="qname" behavior="list of ncname" ? />
</relationshipType>
```

2.1.4 ChannelType

After defining the roles and token, the channel will be defined. The <channelType> is the main mechanism used to achieve interaction. ChannelType makes a point of collaboration between participantType specifying where and how information is exchanged.

Listing 4: ChannelType General Structure

```
<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>
```

2.2 Dynamic Part

<Choreography> carries a set of relationships (interactions between the types of roles) and contains a definition of all variables used to perform the behavior according to the data of the choreography.

Listing 5: Choreography General Structure

```
<choreography
  name="NCName"
  complete="xsd:boolean XPath-expression" ?
  isolation="true" | "false" ?
  root="true" | "false" ?
  coordination="true" | "false"? >
  <relationship type="QName" /> +
  variableDefinitions ?
  Choreography-Notation *
  Activity-Notation
  <exceptionBlock name="NCName">
  WorkUnit-Notation+
  </exceptionBlock> ?
  <finalizerBlock name="NCName">
  Activity-Notation
  </finalizerBlock> *
</choreography>
```

Activities are low-level components of the choreography and used to describe the actual work performed. The activity-notation is used to define these activities

2.2.1 Basic Activity

The main basic activities are <interaction> and <assign>:

- <interaction> : is the basic building block of a choreography, which translates the exchanged information between the collaborating parties and possibly the synchronization of their observable information and values âĂŃâĂŃof the exchanged information changes. It is defined by:
  - channelVariable
  - participate with relationshipType, fromRoleTypeRef and toRoleTypeRef
  - exchange which contains name, informationType/channelType and action

Listing 6: Interaction General Structure

```
<interaction
  name="NCName"
  channelVariable="QName"
  operation="NCName"
  align="true" | "false" ?
  initiate="true" | "false" ? >
  <participate
  relationshipType="QName"
  fromRoleTypeRef="QName" toRoleTypeRef="QName" />
  <exchange
    name="NCName"
    faultName="QName" ?
    informationType="QName" ? | channelType="QName" ?
    action="request" | "respond" >
    <send
      variable="XPath-expression" ?
      recordReference="list of NCName" ?
      causeException="QName" ? />
    <receive
      variable="XPath-expression" ?
      recordReference="list of NCName" ?
      causeException="QName" ? />
  </exchange> *
</interaction>
```

- <assign> : is an activity used to create or change, and then make it available in a role, the value of one or more variables. It can be, also, used to cause an exception to a role.

Listing 7: Assign General Structure

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

2.2.2 Structural Activity

It refers to activities that specify scheduling rules in the choreography. At this level, we distinguish:

- <sequence> : contains one or more Activity-Notation. When the sequence activity is activated, the block of the elements are sequentially activated in the same order as their definitions.

- <parallel> :the control structure, parallel, contains one or several Activity-Notation. The parallel activity completes successfully when all the activities involved carry out work work successfully.
- <choice> : the choice control structure specifies the activity to be performed.

2.2.3 WorkUnit-Notation

<workunit> may prescribe the constraints which must be satisfied to achieve progress and thus perform the actual work in a choreography. It may also prescribe the constraints which preserve the consistency of collaborations commonly performed between the parties. Using workunit, an application can retrieve the errors that are the result of abnormal actions and can also finalize choreography

Listing 8: Workunit General Structure

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

WS-CDL Sample

```
 <?xml version="1.0" encoding="UTF-8" ?>
<package name="BuyerSellerCDL" author="Steve Ross-Talbot"
version="1.0"
xmlns:bs="http://www.pi4tech.com/cdl/BuyerSellerExample-1" >
<description type="description">This is the basic ↩
    BuyerSeller Choreography Description</description>
<informationType name="BooleanType" type="xs:boolean" />
<informationType name="StringType" type="xsd:string" />
<informationType name="RequestForQuoteType" type="bs:↩
    RequestForQuote">
<description type="documentation"> Request for quote message↩
     </description>
</informationType>
<informationType name="QuoteType" type="bs:Quote">
<description type="documentation"> Quote message </↩
    description>
</informationType>
<informationType name="QuoteUpdateType" type="bs:QuoteUpdate↩
    ">
<description type="documentation"> Quote Update Message </↩
    description>
</informationType>
<informationType name="QuoteAcceptType" type="bs:QuoteAccept↩
    ">
<description type="documentation">Quote Accept Message</↩
    description>
</informationType>
<informationType name="CreditCheckType"
type="bs:CreditCheckRequest">
<description type="documentation"> Credit Check Message </↩
    description>
</informationType>
<informationType name="CreditAcceptType" type="bs:↩
    CreditAccept">
<description type="documentation"> Credit Accept Message </↩
    description>
</informationType>
<informationType name="CreditRejectType" type="bs:↩
    CreditReject">
<description type="documentation"> Credit Reject Message </↩
    description>
</informationType>
<informationType name="RequestDeliveryType" type="bs:↩
    RequestForDelivery">
<description type="documentation"> Request Delivery Message ↩
    </description>
</informationType>
```

```
<informationType name="DeliveryDetailsType"
type="bs:DeliveryDetails">
<description type="documentation"> Delivery Details Message ↩
    </description>
</informationType>
<token name="BuyerRef" informationType="StringType" />
<token name="SellerRef" informationType="StringType" />
<token name="CreditCheckRef" informationType="StringType" />
<token name="ShipperRef" informationType="StringType" />\\
<roleType name="BuyerRoleType">
<description type="description"> The Behavior embodied by a ↩
    buyer </description>
<behavior name="BuyerBehavior" />
</roleType>\\
<roleType name="SellerRoleType">
<description type="description"> The behavior embodied by a ↩
    seller </description>
<behavior name="SellerBehavior" />
</roleType>\\
<roleType name="CreditCheckerRoleType">
<description type="description"> The behavior embodied by a ↩
    credit checker service </description>
<behavior name="CreditCheckerBehavior" />
</roleType>\\
<roleType name="ShipperRoleType">
<description type="description"> The behavior embodied by a ↩
    shipper service </description>\\
<behavior name="ShipperBehavior" />
</roleType>\\
<relationshipType name="BuyerSeller">
<role type="BuyerRoleType" />
<role type="SellerRoleType" />
</relationshipType>\\
<relationshipType name="SellerCreditCheck">
<role type="SellerRoleType" />
<role type="CreditCheckerRoleType" />
</relationshipType>\\
<relationshipType name="SellerShipper">
<role type="SellerRoleType" />
<role type="ShipperRoleType" />
</relationshipType>
<relationshipType name="ShipperBuyer">
<role type="ShipperRoleType" />
<role type="BuyerRoleType" />
</relationshipType>
<channelType name="Buyer2SellerChannelType">
<passing channel="2BuyerChannelType" new="true">
<description type="description"> Pass channel to enable ↩
    shipper to talk to buyer </description>
</passing>
```

```
<role type="SellerRoleType" />
<reference>
<token name="SellerRef" />
</reference>
</channelType>
<channelType name="Seller2CreditCheckChannelType">
<role type="CreditCheckerRoleType" />
<reference>
<token name="CreditCheckRef" />
</reference>
</channelType>
<channelType name="2BuyerChannelType" action="request">
<role type="BuyerRoleType" />
<reference>\\
<token name="BuyerRef" />
</reference>
</channelType>
<channelType name="Seller2ShipperChannelType">
<passing channel="2BuyerChannelType">
<description type="description"> Pass channel through to ↩
    shipper </description>
</passing>
<role type="ShipperRoleType" />
<reference>
<token name="ShipperRef" />
</reference>
</channelType>
<choreography name="Main" root="true">
<description type="description"> Collaboration between buyer↩
    , seller, shipper, credit chk </description>
<relationship type="BuyerSeller" />
<relationship type="SellerCreditCheck" />
<relationship type="SellerShipper" />
<relationship type="ShipperBuyer" />
<sequence>
<interaction name="Buyer requests a Quote - this is the ↩
    initiator"\\
operation="requestForQuote" channelVariable="Buyer2SellerC"
initiate="true">
<description type="description">Request for Quote</↩
    description>
<participate relationshipType="BuyerSeller" fromRole="↩
    BuyerRoleType" toRole="SellerRoleType" />
<exchange name="request" informationType="↩
    RequestForQuoteType" action="request">
<description type="description">Requesting Quote</↩
    description>
</exchange>
<exchange name="response" informationType="QuoteType"
action="respond">
```

```
<description type="description">Quote returned</description>
</exchange>
</interaction>
<workunit name="Repeat until bartering has been completed"
repeat="barteringDone = false">
<choice>
<sequence>
<interaction name="Buyer accepts the quote and engages in ↩
    the act of buying"
operation="quoteAccept" channelVariable="Buyer2SellerC" >
<description type="description">Quote Accept</description>
<participate relationshipType="BuyerSeller"
fromRole="BuyerRoleType" toRole="SellerRoleType" />
<exchange name="Accept Quote" informationType="↩
    QuoteAcceptType"
action="request">
</exchange>
</interaction>
<interaction name= "Buyer send channel to seller to enable ↩
    callback behavior"
operation="sendChannel" channelVariable="Buyer2SellerC">
<description type="description"> Buyer sends channel to pass↩
     to shipper </description>
<participate relationshipType="BuyerSeller"
fromRole="BuyerRoleType" toRole="SellerRoleType" />
<exchange name="sendChannel" channelType="2BuyerChannelType"↩
     action="request">
<send variable="cdl:getVariable('DeliveryDetailsC','','')" ↩
    />
<receive variable="cdl:getVariable('DeliveryDetailsC','','')↩
    " />
</exchange>
</interaction>
<assign roleType="BuyerRoleType">
<copy name="copy">
<source expression="true" />
<target variable="cdl:getVariable('barteringDone','','')" />
</copy>
</assign>
</sequence>
<sequence>
<interaction name="Buyer updates the Quote - in effect ↩
    requesting a new price"
operation="quoteUpdate" channelVariable="Buyer2SellerC">
<description type="documentation">Quot Update</description>
<participate relationshipType="BuyerSeller"
fromRole="BuyerRoleType" toRole="SellerRoleType" />
<exchange name="updateQuote"
informationType="QuoteUpdateType" action="request">
</exchange>
```

```
<exchange name="acceptUpdatedQuote"
informationType="QuoteAcceptType" action="respond">
<description type="documentation"> Accept Updated Quote </↩
    description>
</exchange>
</interaction>
</sequence>
</choice>
</workunit>
<interaction name="Seller check credit with CreditChecker"
operation="creditCheck" channelVariable="Seller2CreditChkC">
<description type="description">
Check the credit for this buyer with the credit check agency
</description>
<participate relationshipType="SellerCreditCheck"
fromRole="SellerRoleType" toRole="CreditCheckerRoleType" />
<exchange name="checkCredit" informationType="↩
    CreditCheckType" action="request">
</exchange>
</interaction>
<choice>
<interaction name="Credit Checker fails credit check"
operation="creditFailed" channelVariable="Seller2CreditChkC"↩
    >
<description type="description">
Credit response from the credit checking agency
</description>
<participate relationshipType="SellerCreditCheck"
fromRole="SellerRoleType" toRole="CreditCheckerRoleType" />
<exchange name="creditCheckFails" informationType="↩
    CreditRejectType" action="respond">
</exchange>
</interaction>
<sequence>
<interaction name="Credit Checker passes credit"
operation="creditOk" channelVariable="Seller2CreditChkC">
<description type="description">
Credit response from the credit checking agency
</description>
<participate relationshipType="SellerCreditCheck" fromRole="↩
    BuyerRoleType"
toRole="CreditCheckerRoleType" />
<exchange name="creditCheckPasses"
informationType="CreditAcceptType" action="respond">
</exchange>
</interaction>
<interaction name="Seller requests delivery details"
operation="requestShipping" channelVariable="Seller2ShipperC↩
    ">
```

```
<description type="description"> Request delivery from the ↩
    shipper </description>
<participate relationshipType="SellerShipper"
fromRole="SellerRoleType" toRole="ShipperRoleType" />
<exchange name="sellerRequestsDelivery"
informationType="RequestDeliveryType" action="request">
</exchange>
<exchange name="sellerReturnsDelivery"
informationType="DeliveryDetailsType" action="respond">
</exchange>
</interaction>
<interaction name="Shipper forward channel to shipper"
operation="sendChannel" channelVariable="Seller2ShipperC">
<description type="description"> Pass channel from buyer to ↩
    shipper </description>
<participate relationshipType="SellerShipper"
fromRole="SellerRoleType" toRole="ShipperRoleType" />
<exchange name="forwardChannel" channelType="2↩
    BuyerChannelType" action="request">
<send variable="cdl:getVariable('DeliveryDetailsC','','')" ↩
    />
<receive variable="cdl:getVariable('DeliveryDetailsC','','')↩
    " />
</exchange>
</interaction>
<interaction name="Shipper sends delivery details to buyer"
operation="deliveryDetails" channelVariable="↩
    DeliveryDetailsC">
<description type="description"> Pass back shipping details ↩
    to the buyer </description>
<participate relationshipType="ShipperBuyer"

fromRole="ShipperRoleType" toRole="BuyerRoleType" />
<exchange name="sendDeliveryDetails"
informationType="DeliveryDetailsType" action="request">
</exchange>

</interaction>
</sequence>
</choice>
</sequence>
</choreography>
</package>
```

References

[1] A. Barros, M. Dumas, and P. Oaks. A critical overview of the web services choreography description language (ws-cdl). *BPTrends*, 2005.

[2] G. Decker, H. Overdick, and J. M. Zaha. On the suitability of ws-cdl for

choreography modeling. In *Proceedings of methoden, konzepte und technologien fur die entwiclkung von dienstebasierten informationssystemen (EMISA 2006)*, 2006.

[3] L. Fredlund. Implementing ws-cdl. In *McErlang: a model checker for a distributed functional programming language*, pages 125–136, 2006.

[4] Niels Lohmann, Oliver Kopp, Frank Leymann, and Wolfgang Reisig. Analyzing bpel4chor: Verification and participant synthesis. In *WS-FM*, 2007.

[5] M. Rani, A. Kumar Chawla, and S. Batra. Web service choreography description language (ws-cdl): Goals and benefits. *COIT*, 2006. http://www.rimtengg.com/coit2007/proceedings/pdfs/49.pdf.

[6] B. Schmeling, A. Charfi, and M. Mezini. Composing non-functional concerns in composite web services. In *ICWS*, pages 331–338, 2011.

[7] B. Silver. *BPMN 2.0 Handbook*. Workflow Management Coalition, 2011.

[8] Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, and Arthur H. M. ter Hofstede. Let's dance: A language for service behavior modeling. In *OTM Conferences (1)*, pages 145–162, 2006.