WS-CDL Primer

Date: 17th January 2005
Last Modified: 17th March 2005
Authors: Steve Ross-Talbot, Anthony Fletcher
Version: V0.2

1. Introduction

1.1 Prerequisites

This primer assumes that the reader has the following prerequisite knowledge:

- familiarity with XML (Extensible Markup Language (XML) 1.0 (Second Edition) [XML 1.0], XML Namespaces (Namespaces in XML [XML Namespaces]), WSDL (Web Services Description Language) 1.1 and 2.0;
- some familiarity with XML Schema (XML Schema Part 1: Structures [XML Schema: Structures] XML Schema Part 2: Datatypes [XML Schema: Datatypes]), SOAP (Simple Object Access Protocol) 1.2;
- familiarity with basic Web services concepts such as
- Web service, client, and the purpose and function of a Web service description. (For an explanation of basic Web services concepts, see Web Services Architecture [WS Architecture] Section 1.4 and Web Services Glossary [WS Glossary] glossary. However, note the Web Services Architecture document uses the slightly more precise terms "requester agent" and "provider agent" instead of the terms "client" and "Web service" used in this primer.) Put these in sub-bullets to increase readabililty.

No previous experience with WS-CDL is assumed.

1.2 Structure of this Primer

Section 2 presents an overview of WS-CDL. It segments the language and describes what it can be used for and presents the business benefits that can be gained from using WS-CDL. Finally a methodology is described which provides a guide as to how to build a choreography.

Section 3 presents a hypothetical use case involving a market in which buyers and sellers and supporting roles enact their business. It proceeds step-by-step based on the methodology described in Section 2 through the development of this simple example.

Section 4 introduces more advanced topics that deal with notions of business collaborations and failure as well as complex ordering constraints that can be described in WS-CDL.

Section 5 describes the relationships that WS-CDL has to some of the other standards and technologies. [This includes code generation to Java, .NET and BPEL4WS, the relationship through the use of extensions to Policy based technology and Service Discovery.] Revise according to today's discussion.

1.3 Notational Conventions

The following namespace prefixes are used throughout this document:

<Table>

This specification uses an informal syntax to describe the XML grammar of a WS-CDL document:

- The syntax appears as an XML instance, but the values indicate the data types instead of values.
- Characters are appended to elements and attributes as follows: "?" (0 or 1), "*" (0 or more), "+" (1 or more).
- Elements names ending in ". . ." (such as <element. . ./> or <element. . .>) indicate that elements/attributes irrelevant to the context are being omitted.
- Grammar in bold has not been introduced earlier in the document, or is of particular interest in an example.
- <-- extensibility element --> is a placeholder for elements from some "other" namespace (like ##other in XSD).
- The XML namespace prefixes (defined above) are used to indicate the namespace of the element being defined.
- Examples starting with <?xml contain enough information to conform to this specification; other examples are fragments and require additional information to be specified in order to conform.

An XSD is provided as a formal definition of WS-CDL grammar (see Section 11 of the Web Services Choreography Description Language Version 1.0 W3C Working Draft 17 December 2004).

2. An Overview of WS-CDL

The Web Services Choreography Description Language (WS-CDL) is an XML-based language that can be used to describe the common and collaborative behavior of multiple parties who need to interact in order to achieve some goal. WS-CDL describes this behavior from a global or neutral perspective rather than from the perspective of any one party.

2.1 Using WS-CDL

WS-CDL is not an executable language, hence the term "Description" in it's name. It is a language that can be used to unambiguously describe service collaborations and their protocols within and across domains of control. [Change made to support the information below]

In the case of the former it can be used to describe the internal workflows within a domain that involves multiple end-points/services that constitute collaborative behavior. The value in so doing is to encourage conformance of services to an agreed choreography description and to improve interoperability of services through an agreed choreography description. This is no more than describing a business protocol that defines a collaboration between services. You can think of it as a way of ensuring services are well behaved with respect to the goals that you want to achieve within your domain.

In the case of the latter it can be used to describe the business protocols across domains such as the ordering of message exchanges that govern vertical protocols such as fpML, FIX, TWIST and SWIFT. These protocols have some form of XML data format definition and then go on to describe the ordering of message exchanges using a combination of prose and UML sequence diagrams. What WS-CDL provides in an unambiguous way of describing the ordering of message exchanges and in so doing ensure that the end points that participate in service collaborations based on such vertical standards can be guaranteed to conform to the choreography description. You can think of it as a way to ensure that participants are well behaved with respect to their common goals across domains.

2.2 The Business Case (reword)

WS-CDL can be used to ensure interoperability within and across domains of control to lower interoperability issues, such as downtime, and create solutions within and across domains of control.

WS-CDL can be used to ensure that the total cost of software systems in a distributed environment, within a domain of control and across the world-wide-web is lowered by guaranteeing that the services that participate in a choreography are well behaved on a continuing basis.[Do we say guarantee? What implications does this place on the software produced?]

Both of these benefits translate into more up-time and so increase top line profits and at the same time they translate into less testing time and so reduce cost of delivery which decreases bottom line costs.[This is a primer, not a marketing document. This section could be consolidated or an actual business scenario provided.]

2.3 The Structure of WS-CDL

WS-CDL is a layered language. It provides different levels of expressibility to describe a choreography. These levels are shown below (a more complete picture is provided by the UML description of WS-CDL in the appendix):

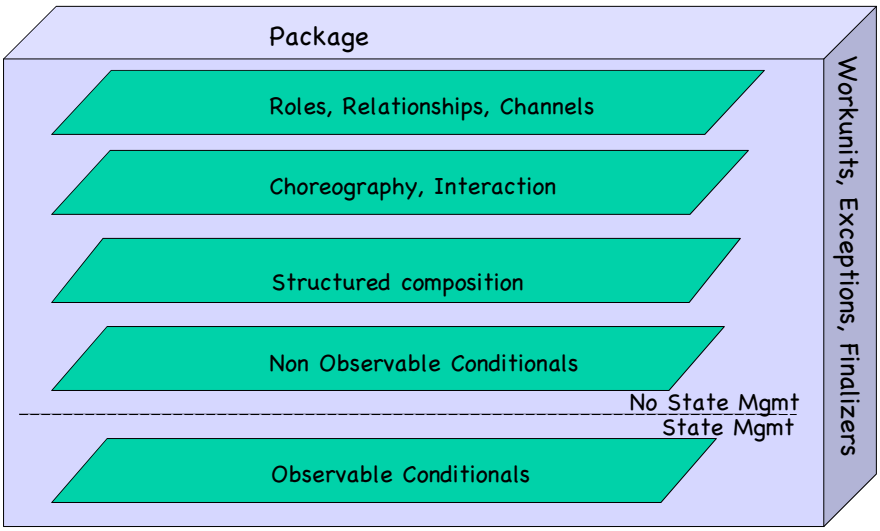


Figure 1: The layering of WS-CDL

At the top most level for any WS-CDL there is a package that contains all other things. All choreographies described in WS-CDL will include as a minimum a set of Roles that are defined as some sort of behavior (i.e. a WSDL description), Relationships between those roles, Channels used by roles to interact and a Choreography block that uses the channels to describe Interaction. What the choreography describes at this level is a

basic set of typed and unambiguous service connections that enable the various roles to collaborate in order to achieve some common goal.

Choreographies that are described using only these features will be uninteresting as there will be few ordering rules. [Don't understand what you are trying to say.]

Adding further ordering rules through Structured composition allows Interactions and Choreographies (which are just logical groupings of interactions) to be combined into sequences, parallel activities and so on.

Adding Non-Observable Conditionals makes it is possible to model branching based on observing changes in the interactions that occur – for example one might observe an exchange between a buyer and a seller which is said to be terminated when a "completed" interaction is observed.

[Is there a mismatch between the previous paragraph about non-observable and this one with observable? Doesn't flow very well.] At this point it is not necessary to perform any explicit state management at the roles that are interacting because we have not needed to express any observable conditions. By this we mean that none of roles used in choreographies at this level have any notion of shared state, rather they observe interactions that are visible and use the observations to determine their state with respect to the other roles.

Some business protocols are defined with specific business rules visible. These constitute shared knowledge between the roles concerned, for example we might terminate an order completion between a buyer and a seller when we calculate that the items delivered match the original order. The business rule in this example is the shared constraint that buyer_quantity equals completion_quantity. At some level the roles must have some shared knowledge of both variables and their values. When business rules of this nature become part of the business protocol such Observable Conditionals can be added into a choreography and this now implies state management is needed.

State management requires an amount of machinery to ensure that state is known globally when needed or at least between roles when needed. The machinery required to deliver this in practice is some coordination mechanism that will ensure data is delivered to all roles that require it.

2.4 Methodology

Given that WS-CDL is a language that described interaction between roles it is natural to look first for the roles that will interact. This is the starting point for any choreography. The methodology described herein documents a process that can be used to guide the user in creating a choreography.

1. Identify the role that will interact
2. Defining the relationships
   - between the roles that will communicate
3. Define the channels
   - that will be used to connect the roles together
   - directionality: Who receives and who initiates
4. Group the roles into participants
5. Sketch the choreographies (a grouping of high level logical interactions)
6. Define the interactions
   - Channel variables, Information Types
7. Define state management
   - Records, variables
8. Refine the choreographies (by grouping into finer grain interactions)
9. Work Units, Finalization

This process is not the only process that can be used to define choreographies but we have found it generally consistent in using the WS-CDL to describe complex business protocols.[Break bullets to simple and advanced functions.]

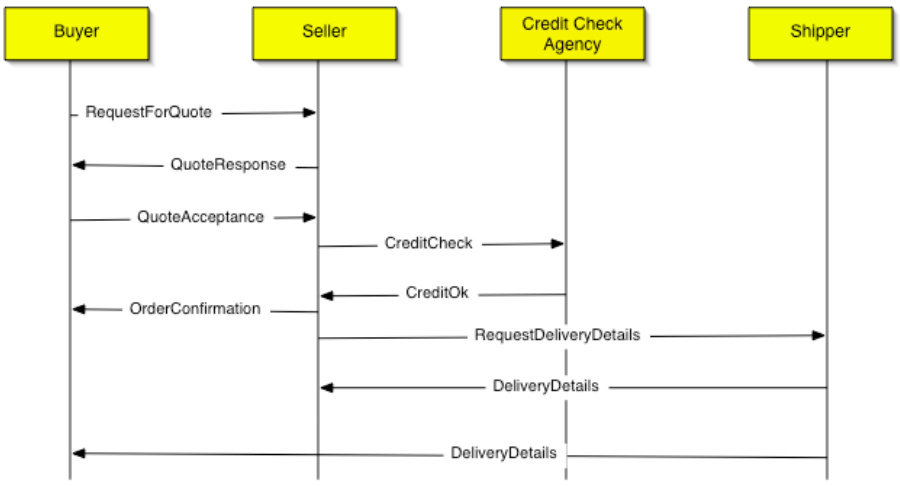
3. Getting Started with WS-CDL

In order to understand WS-CDL is best to illustrate it through the use of an example. In this section we shall introduce an example and use it throughout the rest of primer building on it to illustrate different parts of WS-CDL. The Appendicies have the full listing of the various WS-CDL encodings of the example as well as a url to the WS-CDL descriptions. In all cases the WS-CDL descriptions have been tested against at least one implementation of WS-CDL having been constructed in a validating editor.

3.1 An Example

The example that we use concerns the collaborative behavior of a buyer, a seller, a credit check agency and a shipper. In this example the buyer interacts with the seller to determine a price. When a price is acceptable to the buyer the buyer orders the relevant goods based on this price where upon the seller checks their credit worthiness and if this is acceptable requests a delivery date from the shipper. In our example the shipper communicated directly with the buyer once an agreed delivery date has been achieved and informs the buyer of the delivery details.

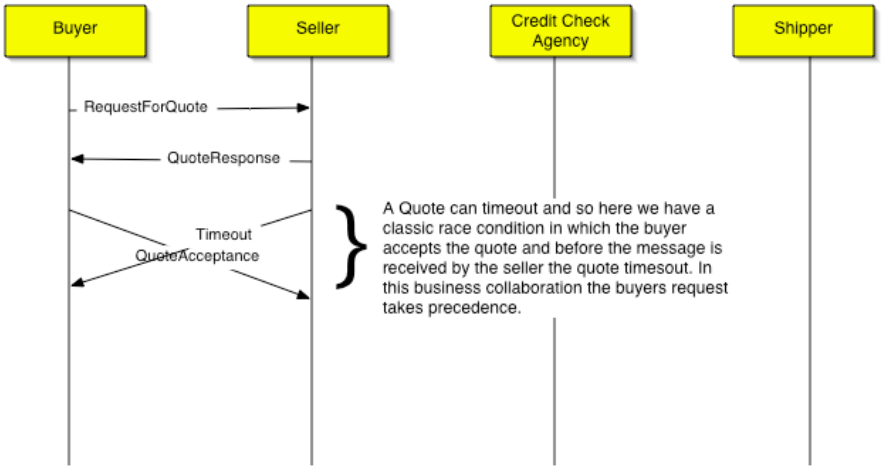
The example is further illustrated by means of a number of sequence diagrams below:



Normal Collaboration

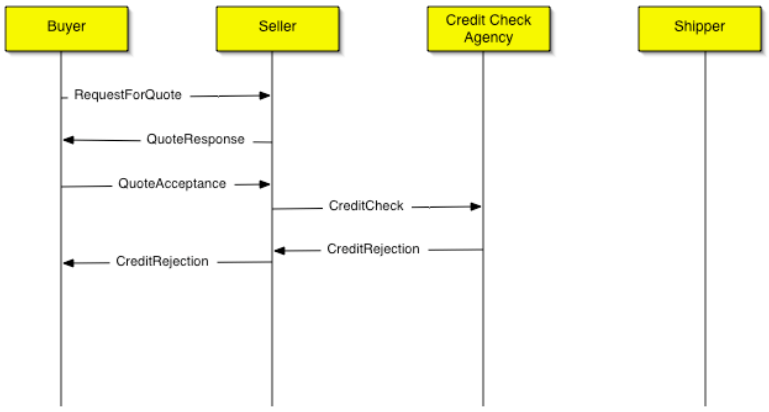
The Normal Collaboration, shown in Figure 2, shows the buyer requesting a quote and the seller responding with a quote. The buyer then accepts the quote, which is akin to placing the order. As a result the seller checks

the buyers credit rating. Because the buyers credit rating is ok the seller then confirms the order with the buyer and requests from the shipper delivery details which are passed back to the seller by the shipper. The shipper will have picked up all that is necessary from the shipper, who received it from the buyer as part of the order placement, all of the details necessary to communicate directly with the buyer so that delivery details can be passed back from the shipper to the buyer.



Quote Timeout Collaboration

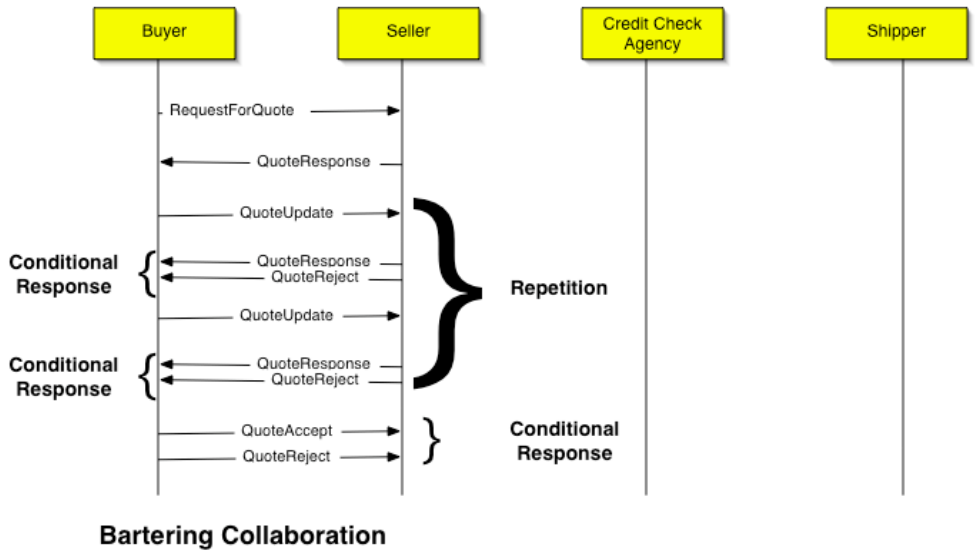
The Quote Timeout Collaboration, shown in Figure 3, shows the buyer requesting a quote, the seller sending back a quote response that has a timeout associated with the quote. If the buyer fails to act on the quote in time (before the timeout elapses) the buyer may not honor the quote. In the scenario presented we show the opportunity for the buyer to accept the quote just as the seller decides that the quote has timed-out. This demonstrates a classic race condition between the buyer and the seller.



Credit Rejection Collaboration

Figure 4 shows a credit check rejection for the buyer. After the buyer requests the quote and the seller responds with a quote and then the buyer accepts the quote the seller then checks the credit rating for the buyer and in this case the credit check agency determines that the credit rating is low and so send back a credit rejection to the seller who in turn sends a credit rejection to the buyer and terminating the collaboration.

The final scenario to introduce is that of the bartering collaboration. This is shown in Figure 5 below.



Bartering Collaboration

In this collaboration the buyer requests a quote from the seller who responds with a quote. Thereafter the buyer may request an updated quote, filling in a desired price and quantity, from the seller. The seller may respond by rejecting the quote in which case the buyer may try again, or the seller may accept the quote by sending a quote response message back to the seller. The seller in receipt of either a quote reject or a quote response from the seller may accept the quote (and by so doing enter into the act of buying with the seller) or may request an updated quote or indeed may reject the quote or simply do nothing at all – which allows the quote to timeout.

We have used a heavily annotated form of sequence diagram to describe the business collaboration protocol necessary for the buyer, seller, credit agency and shipper to go about their business. WS-CDL is very much designed to enable the entire business collaboration protocol to be described in an unambiguous manner. We hope that this becomes self evident to the reader as we walk through constructing the WS-CDL description for this example.

3.2. Interactions, Roles, Tokens and Channels

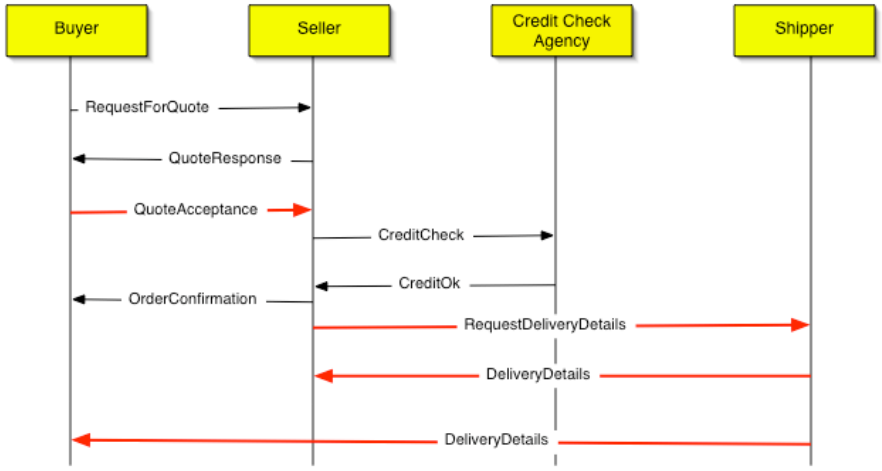
In this section we shall introduce the fundamental concept of an interaction, which underpins WS-CDL and in so doing define the roles, tokens, channels, relationships, participants and variables necessary to use an interaction in order to describe our business protocol in the example.

3.2.1 Defining some interactions

An interaction is the realization of a collaboration between roles or participants. We shall explain the difference between roles and participants later on for now we can consider them as synonomous.

A collaboration is some sort of message exchange between the swim lanes of a sequence diagram. We shall focus on a portion of the normal collaboration between a buyer and seller to illustrate the use of interactions in WS-CDL.

In the diagram below we show the same normal collaboration described earlier but have changed the colours of the relevant arrows to red. These red arrows are what we shall model with our interactions.



Normal Collaboration

In the example when the buyer accepts a quote it does some extra things so that the seller, on behalf of the buyer, can pass some sort of contact details to a third party who can contact the buyer to inform the buyer of the delivery details. The buyer send to the seller a quote acceptance. This exchange of information is further annotated to ensure that the collaboration includes the details for a third party to contact the buyer with the delivery details. When the seller receives the necessary

information for quote acceptance the seller passes this information to the shipper, as part of a collaboration, to determine suitable delivery details. The shipper then uses this extra information to inform the buyer directly. It is this collaboration between the buyer, seller and shipper than we shall use to illustrate the use of interactions.

The WS-CDL fragment for these red arrows is shown below:

```
<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 new channel to pass on 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>

<interaction name="Seller requests delivery details - passing channel for buyer and shipper to interact"
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="2BuyerChannelType" 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>
```

3.2.2 Defining the roles

There are 4 roles that are played out in the example. These are the "buyer" the "seller" the "credit agency" and the "shipper".

```
<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 </description>
  <behavior name="CreditCheckerBehavior" />
</roleType>

<roleType name="ShipperRoleType">
  <description type="description">The behavior embodied by a shipper service</description>
  <behavior name="ShipperBehavior" />
</roleType>
```

A role in WS-CDL is a named behavior and it is clear from the example that the roles we have identified have behavior relative to one another. This is what collaboration is all about; identifying common interaction between peers.

In this example we shall assume that none of the roles have any web services defined for them and so we can omit the interface attribute. Later on we shall recast the example based such that one or more of the roles do have existing web services defined for them.

The roles identified are shown above in a WS-CDL fragment.

The abstract syntax for roles is shown below:

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

3.2.3 Defining the tokens

The reference token refers to a service reference that is a url to the web service. In this context a token defines an alias to the web service so that we can treat refer to it by a shorter name. In our example we will not reference any web service url so we define it as a StringType.

<???Check with Gary about this 'cos this doesn't feel right at all???>

For our example we would define the tokens needed for our channels as follows:

```
<token name="BuyerRef" informationType="StringType" />
<token name="SellerRef" informationType="StringType" />
<token name="CreditCheckRef" informationType="StringType" />
<token name="ShipperRef" informationType="StringType" />
```

The abstract syntax for defining a token is as follows:

```
<token  name="ncname"  informationType="qname" />
```

3.2.4 Defining the channels

Finally, having defined our roles and tokens we are in a position to define our channels.

The abstract syntax of a channel definition is provided below and we shall walk through the steps we need to take to fully define the channels for the example presented earlier.

```
<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> >
```

The roleType name declares who is the service provider for the channel. For example we might have a channel between the buyer and the seller that enables the collaboration between the buyer and seller in the sequence diagrams. In this case the seller is playing the role of service provider and the buyer the role of client and so a channel that we might name “Buyer2SellerChannelType” would have a role of “SellerRole”.

The full definition for the Buyer2SellerChannelType is defined below:

```
<channelType name="Buyer2SellerChannelType">
        <passing channel="2BuyerChannelType" new="true">
                <description
                        type="description">Able to pass channel to enable shipper to talk to
                </description>
        </passing>

        <role type="SellerRoleType" />
        <reference>
                <token name="SellerRef" />
        </reference>

</channelType>
```

The rest of the Channel Types for the example are defined below:

```
<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>
```

In our example we have two sorts of channel types defined. The Buyer2SellerChannelType and the Seller2ShipperChannelType include “passing channel” details, whereas the Seller2CreditCheckChannelType and the 2BuyerChannelType do not have this attribute.

In our example when the buyer decides to accept the quote two things happen. Firstly the buyer sends a message to the seller accepting the quote and then sends a further message to the seller with the details of a channel that it passes to the seller. The seller does a similar thing when it requests delivery details from the shipper; sending the request for delivery details and then sending the channel it received from the buyer on to the shipper. This is all done so that the buyer can receive delivery details back from a third party that it has no knowledge of. Channel passing is how we achieve this and to do it we add details to a channel allowing the channel to pass other channels of a particular type. In our example the type of channel to be passed is the “2BuyerChannelType”.

3.3 Defining the relationships and participants.

Once we have some roles defined we can define the relationships. In WS-CDL a relationship declares an intention to interact between two roles. In a sequence diagram this is akin to any two of the actors (check the name here) who have a connecting arrow in any direction. So in our example we have relationships between the

- buyer and seller
- seller and credit agency
- seller and shipper
- buyer and shipper

Defining these in WS-CDL would look like the following WS-CDL fragment:

```
<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>
```

A relationship comprises a name and two role types. The first role type defines the from role and the second the to role. Hence the ShipperBuyer role in the example has the ShipperRoleType as the first role and the BuyerRoleType as the second role to indicate directionality.

The abstract syntax for relationships is defined as follows:

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

3.4 Choreographies, Sequences, Choices and Workunits

3.4.1 Defining a choreography

3.4.2 Defining a sequence (simple ordering)

3.4.3 Defining a workunit with repetition

3.5 Binding to WSDL

4 Advanced topics

4.1 Modeling business exception in WS-CDL

4.1.1 Defining exceptions

4.1.2 Modeling exceptions as messages

4.2 Modeling compensations in WS-CDL

4.2.1 Defining finalizers

4.3 Modularization in WS-CDL

4.3.1 Splitting into choreographies and sub-choreographies

4.3.2 Performing a choreography

4.4 Data driven collaboration

4.4.1 Using synchronization in Workunits

Appendix 1 – Simple WS-CDL encoding of the example.

```
<?xml version="1.0" encoding="UTF-8" ?>
<package name="BuyerSellerCDL" author="Steve Ross-Talbot" version="1.0"
targetNamespace="www.pi4tech.com/cdl/BuyerSeller" xmlns="http://www.w3.org/2004/12/ws-chor/cdl"
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="description">Request for quote message</description>
    </informationType>

    <informationType name="QuoteType" type="bs:Quote">
        <description type="description">Quote message</description>
    </informationType>

    <informationType name="QuoteRejectType" type="bs:QuoteReject">
        <description type="description">Quote reject message</description>
    </informationType>

    <informationType name="QuoteUpdateType" type="bs:QuoteUpdate">
        <description type="description">Quote Update Message</description>
    </informationType>

    <informationType name="QuoteAcceptType" type="bs:QuoteAccept">
        <description type="description">Quote Accept Message</description>
    </informationType>

    <informationType name="CreditCheckType" type="bs:CreditCheckRequest">
        <description type="description">Credit Check Message</description>
    </informationType>

    <informationType name="CreditAcceptType" type="bs:CreditAccept">
        <description type="description">Credit Accept Message</description>
    </informationType>

    <informationType name="CreditRejectType" type="bs:CreditReject">
        <description type="description">Credit Reject Message</description>
    </informationType>

    <informationType name="RequestDeliveryType" type="bs:RequestForDelivery">
        <description type="description">Request Delivery Message</description>
    </informationType>

    <informationType name="DeliveryDetailsType" type="bs:DeliveryDetails">
        <description type="description">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" />

    <variableDefinitions>
        <variable name="Buyer2SellerC"
            channelType="Buyer2SellerChannelType"
            roleTypes="BuyerRoleType">
            <description type="description">
                Principle channel used to enable interaction between buyer
                and seller for price requests, price confirms and orders
            </description>
        </variable>
        <variable name="Seller2ShipperC"
            channelType="Seller2ShipperChannelType"
            roleTypes="SellerRoleType">
            <description type="description">
                Seller to shipper channel - used to pass a channel to effect
                interaction with the buyer
            </description>
        </variable>
        <variable name="Seller2CreditChkC"
            channelType="Seller2CreditCheckChannelType"
            roleTypes="SellerRoleType">
            <description type="description">
                Seller to Credit Check Channel used to check credit for buyers to
                determine if we do business with them
            </description>
        </variable>
        <variable name="DeliveryDetailsC"
            channelType="2BuyerChannelType"
            roleTypes="BuyerRoleType SellerRoleType ShipperRoleType" />
            <description type="description">
                Channel created by the buyer to pass to third parties so that
                They can communicate with the buyer without have linkage
            </description>
        </variable>
        <variable name="barteringDone"
            informationType="BooleanType"
            roleTypes="BuyerRoleType SellerRoleType">
            <description type="description">Has Bartering Finished flag</description>
        </variable>
    </variableDefinitions>
```

```
<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>
          <silentAction roleType="BuyerRoleType">
              <description type="description">Do nothing - let the quote timeout</description>
          </silentAction>

          <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="description">Quot Update</description>
                  <participate relationshipType="BuyerSeller"
                      fromRole="BuyerRoleType" toRole="SellerRoleType" />
                  <exchange name="updateQuote" informationType="QuoteUpdateType"
                      action="request">
                  </exchange>
              </interaction>
```

```
                    <choice>
                        <interaction name="Seller rejects the updated Quote"
                            operation="quoteReject" channelVariable="Buyer2SellerC">
                            <description type="description">Quote Reject</description>
                            <participate relationshipType="BuyerSeller"
                                fromRole="SellerRoleType" toRole="SellerRoleType" />
                            <exchange name="rejectQuote"
                                informationType="QuoteRejectType" action="respond">
                                <description type="description">Reject Updated Quote</description>
                            </exchange>
                        </interaction>

                        <interaction name="Seller accepts the updated Quote"
                            operation="quoteAccept" channelVariable="Buyer2SellerC">
                            <description type="description">Quote Reject</description>
                            <participate relationshipType="BuyerSeller"
                                fromRole="SellerRoleType" toRole="SellerRoleType" />
                            <exchange name="acceptUpdatedQuote" informationType="QuoteAcceptType"
                                action="request">
                                <description type="description">Accept Updated Quote</description>
                            </exchange>
                        </interaction>
                    </choice>
                </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="2BuyerChannelType" 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>
```