Choreography

Dancers interact with one another with no overall concept of direction.
"There is no conductor."

**Orchestration**: assumes there is a central "conductor" who runs all processes.

**Choreography**: describes a transaction without the assumption of central control.
Understanding choreography

In BPMN and BPEL, subprocesses only exist inside individual business entities.
A choreography is a kind of subroutine that encompasses multiple entities.
It is more like a "dance" than a "procedure", in the sense that all entities "participate".
Need for choreography

A WSDL description describes how to call a service, but not in what order!

A **choreography** describes the order in which to make various calls to a service to complete a transaction...

...without proscribing the global order of all transactions (as in BPEL).
Example: login to SIS

Two participants: user and SIS

Dance steps:

User requests SIS page.
SIS prints page on browser.

Repeat until success:

User enters Login name and Password and presses "login".
SIS checks Login name and Password and succeeds if correct.
In this case, the dance "ends" successfully.
prints an error message and fails if the form is too old.
In this case, user presses "back" and "refresh" and tries again.
prints an error message if SIS is unavailable.
In this case, user ends session and tries later.
prints an error message if the password and login don't match.
In this case, user presses "back" and "refresh" and tries again.

End repeat.

This example:
Involves multiple entities
Has multiple failure conditions.
Results in a predictable state if successful.
Choreography concepts

**choreography**: a set of instructions whereby multiple entities interact.

**participant**: an interacting entity within a choreography.

**role**: the type of a participant, expressed as an interface.

**interaction**: a series of steps in which two participants interact, can contain an exchange, sequence, parallel, or workunit.

**exchange**: a step at which data passes between participants.

**sequence**: a sequence of exchanges or other actions, that occur in order.

**parallel**: a set of exchanges that all happen at once.

**workunit**: a repeated conditional set of exchanges, can contain exchange, sequence, parallel or other workunit. Can emulate if, while, wait until, and repeat until constructions.

**choice**: a set of alternatives, one of which happens.

**assign**: assign a value to a variable.
Example: login to SIS

Two participants: user and SIS

<interaction>
  <workunit>Repeat until success:
    <sequence>
      <exchange>User requests SIS page.</exchange>
      <exchange>SIS prints page on browser.</exchange>
      <exchange>User enters Login name and Password and presses "login". </exchange>
      <choice>
        <exchange>SIS prints an error message and fails if the form is too old. </exchange>
        <exchange>SIS prints an error message if SIS is unavailable. </exchange>
        <exchange>SIS prints an error message if the password and login don't match. </exchange>
        <exchange>SIS checks Login name and Password and succeeds if correct. </exchange>
      </choice>
    </sequence>
  </workunit>
</interaction>
Outline of a choreography

<choreography>

<relationship type='...'/>

<variableDefinitions>

...

</variableDefinitions>

... sequence, parallel, interaction, workunit, ...

</choreography>
Describe interactions/precedences.
"Like an arrow" in a BPEL or BPMN diagram.

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

... 

<choreography ...>
  <relationship type="tns:Buyer2Seller"/>
</choreography>

Pasted from <http://www.w3.org/TR/wsd-cdl-10-primer/>
Binding and state

A choreography is a **description of coordinated state.**
Can accumulate state on each side of a transaction.
State is represented via **state variables.**
These are defined **outside** the choreography.
Packages

A choreography is contained inside a package. Package contains

- informationType declarations.
- variable declarations.
- token declarations.
- tokenLocator declarations.
- roleType declarations.

The choreography. A package is a declaration scope: declarations don't cross packages.
Roles and roleTypes

A role describes one side of a set of exchanges. A roleType describes what that side should be capable of doing, e.g., its interface.

Example:

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

...```

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

Pasted from <http://www.w3.org/TR/ws-cdl-10-primer/>

Normally,

SellerBehaviorInterface is defined via WSDL, e.g., the capabilities of the "SellerBehaviorInterface".
A variable is something that can be set through the assign action.
A token is a read-only part of a message.
A tokenLocator describes an algorithm for fetching a token.
InformationType

A declaration of the kind of information in an exchange. Often defined through WSDL. Accessed via "tokens".

Example:

```
<informationType name="QuoteRequestType"
   type="primer:QuoteRequestType">
   <description type="documentation">
       Quote Request Message
   </description>
</informationType>
```

Pasted from <http://www.w3.org/TR/ws-cdl-10-primer/>

QuoteRequestType: a local typename.
primer:QuoteRequestType: a WSDL Xschema type.
Tokens and tokenLocators

A token is a part of a message or variable.
Read-only.
Might be the same for different kinds of messages.

Example:

```xml
<token name="id" informationType="tns:IdentityType"
  <description type="documentation">
    Identity token
  </description>
</token>
...
<tokenLocator tokenName="tns:id"
  informationType="tns:QuoteRequestType"
  query="/quote/@id">
  <description type="documentation">
    Identity for QuoteRequestType
  </description>
</tokenLocator>
```

Pasted from <http://www.w3.org/TR/ws-cdl-10-primer/>

tns: the nameSpace; our namespace for the example.
tns:id: our token called "id".
tns:QuoteRequestType: the type of data in which it lives.
/quote/@id: an XPATH determining the location of the token (in the id attribute of the quote XML element).
Variable assignment

During a sequence, one can assign a value to a variable.

Example:

```xml
<informationType name="Boolean" type="xsd:boolean">
  <description type="documentation">
    Boolean type for use in loop control
  </description>
</informationType>

...<variableDefinitions>
  <variable name="barteringDone">
    <description type="documentation">
      Variable used to control loop exit.
    </description>
    <variableDefinitions>
      ...<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>
  </variable>
</variableDefinitions>

Pasted from <http://www.w3.org/TR/wsd-cdl-10-primer/>
Four kinds of sequencing

sequence: a serial stream of exchanges, etc.
parallel: a parallel stream.
choice: select one of several options.
workunit: if/while/repeat...until.
Encompasses several kinds of conditional execution

Example:
<workunit guard='G' block='B' repeat='R'>
...
</workunit>

Uses

"If" statement
<workunit guard='expression' block='false()'>
...actions...
</workunit>

"wait until" statement
<workunit guard='expression' block='true()'>
...actions...
</workunit>

"repeat...until" statement:
<workunit guard='true()' repeat='expression'>
...actions...
</workunit>

"wait, then repeat"
<workunit guard='expr1' repeat='expr2'
block='true()'>
actions
</workunit>
block='true()'>
   ... actions...
</workunit>

start while
repeat while true,
Blocking

Workunit (G) (R) (B is True)
   Body

Where
   G => guard condition
   R => repeat condition
   B => blocking attribute
   Body => CDL activities within the work unit

A typical order of evaluation is as follows:

   (G) Body (R G) Body (R G) Body

With respect to a G then the G is only evaluated when the variables are available and evaluate to True and otherwise we wait at the guard condition. Thus the Body after the first G only gets executed when G is True. Or put another way Body is primed ready for action and then is executed when G evaluates to True.

   IF G is unavailable or evaluates to False THEN it equates to:

   when (G) {
       Body
   } until (!R)

   IF G is always True THEN it equates to:

   repeat {
       Body
   } until (!R)

   IF R is always False THEN it equates to:

   when (G) {
Pasted from <http://www.w3.org/TR/ws-cdl-10-primer/>
A variable corresponding to communication between roles.

```xml
<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>
...

<variable name="Buyer2SellerC">
  channelType="tns:Buyer2SellerChannel"
  roleTypes="tns:BuyerRole tns:SellerRole"
  <description type="documentation">
    Channel Variable
  </description>
</variable>
...

<interaction name="QuoteElicitation">
  operation="getQuote"
  channelVariable="tns:Buyer2SellerC"
  ...
</interaction>
```

Pasted from <http://www.w3.org/TR/ws-cdl-10-primer/>
Choreography example

Choreography example

<interation name="QuoteElicitation"
operation="getQuote"
channelVariable="tns:Buyer2SellerC">
<description type="documentation">Quote Elicitation</description>

<participate relationshipType="tns:Buyer2Seller"
fromRoleTypeRef="tns:BuyerRole"
toRoleTypeRef="tns:SellerRole"/>

<exchange name="QuoteRequest"
  informationType="tns:QuoteRequestType"
  action="request">
  <description type="documentation">Quote Request Message Exchange</description>
  <send variable="cdl:getVariable('quoteRequest','','')"/>
  <receive variable="cdl:getVariable('quoteRequest','','')"/>
</exchange>

<choose>
  <exchange name="QuoteResponse"
    informationType="tns:QuoteResponseType"
    action="respond">
    <description type="documentation">Quote Response Message Exchange</description>
    <send variable="cdl:getVariable('quoteResponse','','')"/>
    <receive variable="cdl:getVariable('quoteResponse','','')"/>
  </exchange>
  <exchange name="QuoteResponseFault"
    informationType="tns:QuoteResponseFaultType"
    action="respond"
    faultName="InvalidProductFault">
    <description type="documentation">Quote Response Fault Exchange</description>
    <send variable="cdl:getVariable('faultResponse','','')"/>
    <receive variable="cdl:getVariable('faultResponse','','')"/>
  </exchange>
</choose>

</interaction>

Pasted from <http://www.w3.org/TR/ws-cdl-10-primer/> and edited (a bit).
Choreographies have **participants**, whose behavior is specified by **roleTypes**, in terms of **interfaces**, which are defined (optionally) via WSDL. Communication is described via **channel variables**, which have **channelTypes**, which are defined by specifying **tokens** to extract, via **tokenLocator** declarations.

State is represented by **variables**, which are defined through **variableDeclarations**, where types are **XSchema**, and **assign** controls their values. Program flow is represented by **sequences**, **parallel** execution, **choice**, and **workunits**, which emulate common control methods.
The dream of BPEL: an engine implements your business based upon the BPEL specification, interpreted as a program.

Problem with BPEL: very difficult to "reuse" sections of BPEL, because they rely upon local state that might not be relevant except in the specific BPEL.

The dream of WS-CDL: define best practices for interacting with specific services, independent of the application!

WS-CDL is not intended as an exhaustive description of a process, but rather, a more comprehensive description of "how to use" a service than its WSDL.