

**ORACLE<sup>®</sup>**

# Aggregating Web Services:

## Choreography and WS-CDL

**Nickolaos Kavantzias, Web Services Architect**

**Designer and Lead Editor of WS-CDL**

Oracle Corporation, April 5 2004

**ORACLE**

# Agenda

- Web Services Technology Stack
- Aggregating Web Services
  - Orchestration and BPEL4WS
  - Choreography and WS-CDL
- Collaboration Protocols
  - Business Protocols
  - Business Protocol Requirements
  - Modeling Collaboration Protocols
- WS-CDL in Detail

# Web Services

- Organizations are developing new solutions and integrating existing diverse and discrete applications distributed within or across their trusted domain in an effort to improve productivity and reduce operating costs
- Web Services platform acts as the ubiquitous network fabric through which newly built and existing applications cooperate by exchanging information seamlessly, and unconstrained by hardware, operating systems platforms or programming languages

# Current Web Services Technology Stack

- Core communication framework bridging heterogeneous computational models
  - Distributed, Loosely-coupled, stateless
  - Exchange of type checked information



# Emerging Web Services Technology Stack

**Business Collaboration Language: Web Services  
Choreography Description Language (WS-CDL)**

**Business Process Languages:  
Business Process Execution Language (BPEL4WS)**

**Reliable  
Messaging**

**Security**

**Transaction**

**Coordination**

**Service  
Aggregation**

**Quality  
of Service**

**UDDI**

**WSDL**

**SOAP**

**XML, Encoding**

**HTTP, IIOP, JMS, SMTP**

**Discovery**

**Description**

**Messaging**

**Transport**

# Aggregating Web Services

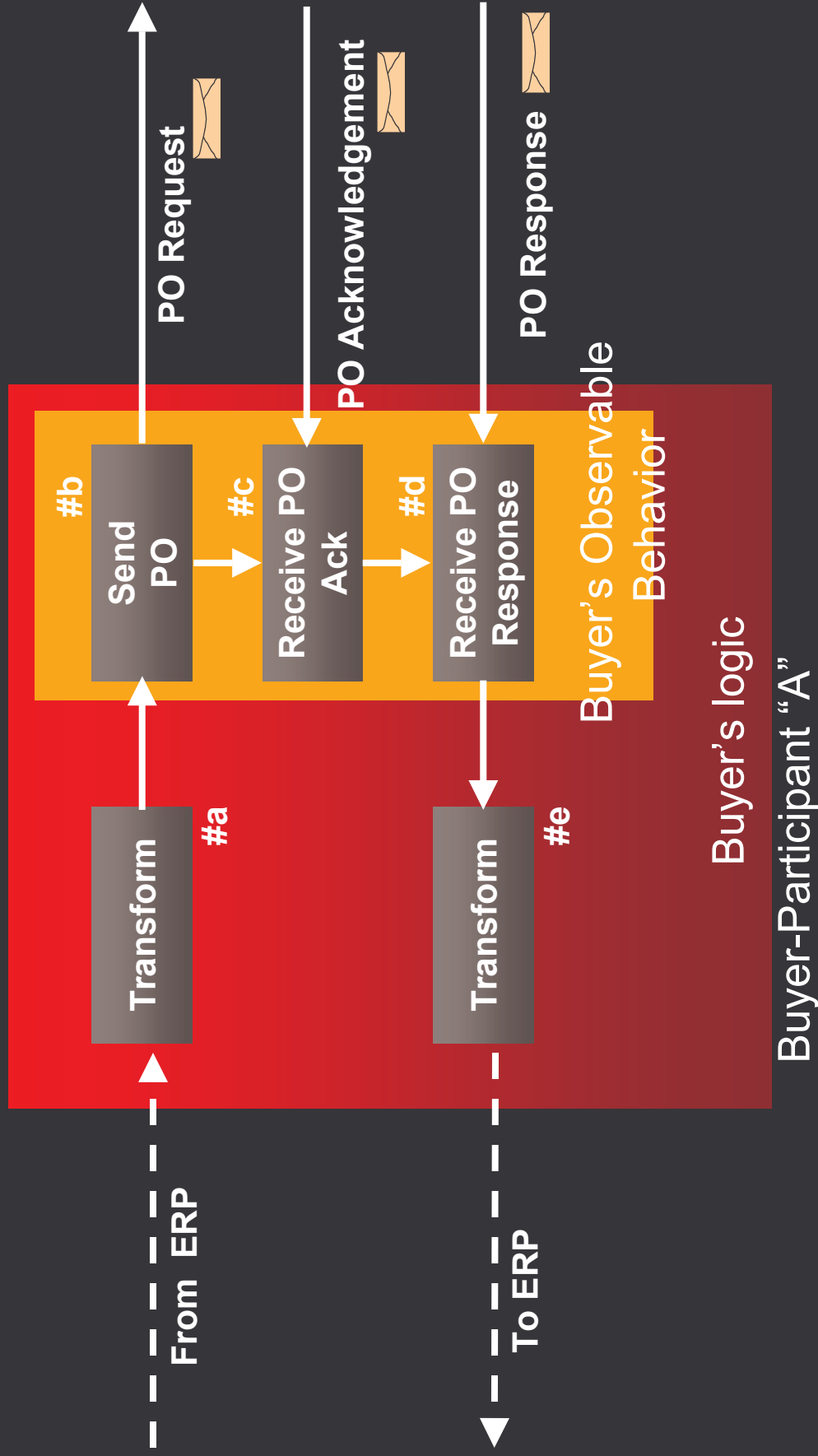
- The inherent value of WS is realized by carefully planning how the various applications are modeled and how they interact with one another and with humans, when aggregated together
- The operational semantics for accomplishing these goals are expressed through:
  - Choreography & Orchestration
  - Choreography complements Orchestration
    - Choreography and Orchestration standards needed for next generation Web Services platform

# Orchestration and BPEL

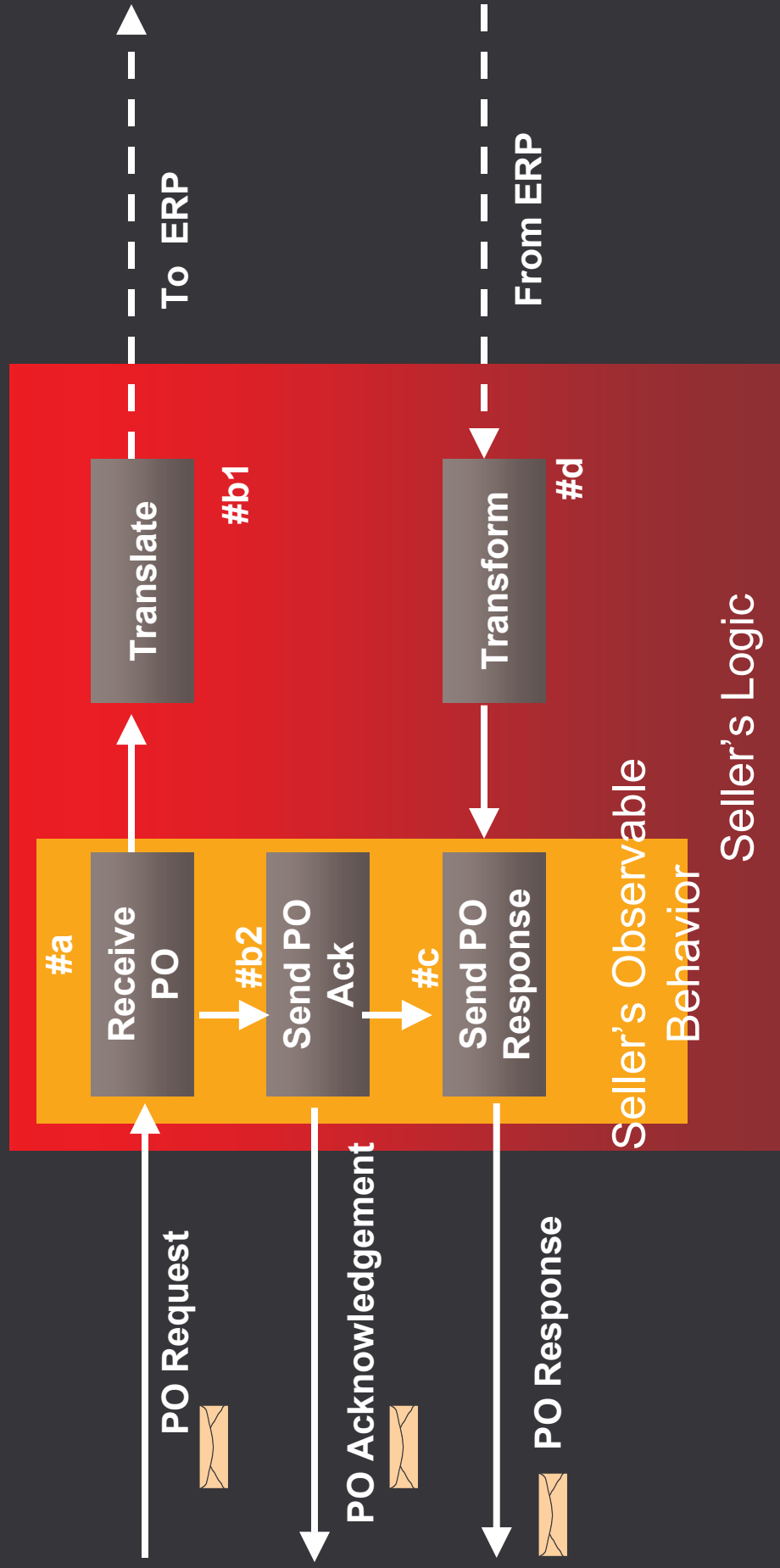
- Orchestration focuses on the behavior of *a single participant*
  - Hub/spoke based model, where a controller residing at a single location in a distributed system, locally enforces the progress of a process by following its definition
- BPEL describes the computational logic of *a participant*
  - control-flow (conditionals, sequencing, parallelism, loops)
  - observable behavior of a participant, from the participant's own viewpoint
  - variables and constructs for manipulating them
  - event handlers
  - timeout management
  - exception and compensation handlers
- Executable-BPEL/Abstract-BPEL
  - They both possess the same computational expressiveness but with different syntactic rules, allowing specifying less process definition information when using AbsBPEL compared to ExecBPEL



# Orchestration: Buyer's Logic, with its Observable Behavior



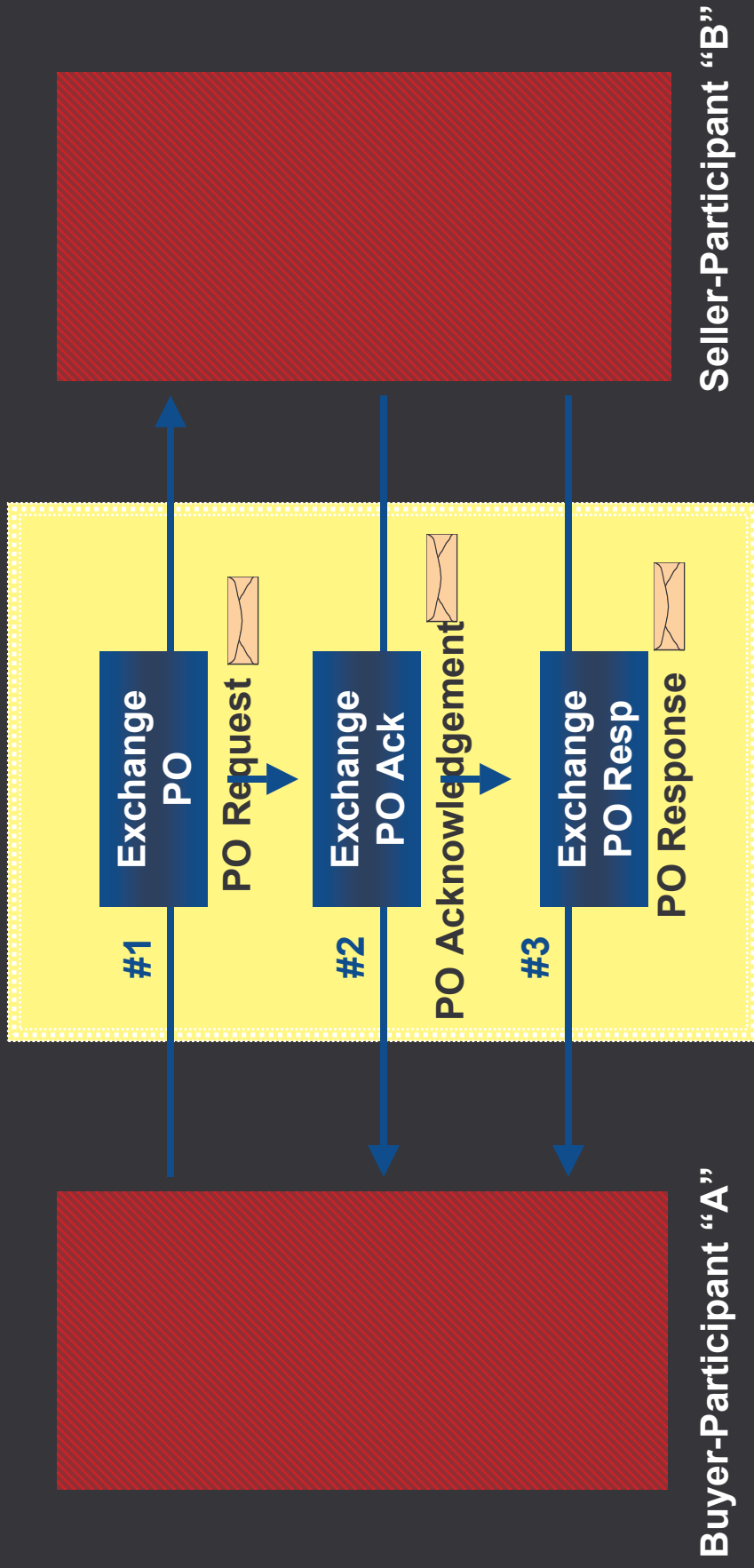
# Orchestration: Seller's Logic with its Observable Behavior



# Choreography

- Choreography is concerned with global, multi-party, peer-to-peer collaborations interoperable between any type of application components, regardless of the supporting platform or programming model used, being distributed within or across an organization's trusted domain
  - Participants act as peers, interacting in long-lived, stateful and coordinated fashion
- Choreography does not depend on a centralized controller

# Choreography: Buyer and Seller Collaborate



Choreography of Buyer's & Seller's Collaboration

ORACLE

# WS-Choreography Description Language (WS-CDL)

- Choreography defines the common observable behavior of two or more participants, focusing on a global, participant agnostic viewpoint; where exchanges of information, with potential information alignment, occur when jointly agreed, information driven reactive rules are satisfied
- WS-CDL is a language in which a Choreography description is specified
  - Initially designed by Oracle and then submitted into the W3C Choreography WG in September 2003
  - Standardization underway in the W3C Choreography WG with first Working Draft published in April 2004

# WS-CDL Usage

- Model a complex business protocol, such as Order Management, enabling interoperability between any type of application components, regardless of the supporting platform or programming model used
- Generate computational logic of an individual collaborating participant
  - ie. BPEL, Java, C#

# WS-CDL Formalisms (1)

- *Global Model formalism* [Nickolaos Kavantzas, work in progress]
  - Based on a variant of pi-calculus [R. Milner, J. Parrow, D. Walker], the Explicit Solos calculus [P. Gardner, C. Laneve, L. Wischik], allows modeling a system from a global viewpoint

## Syntax:

Inf set  $N$  of names  $x, y, u$  and literals

$\mathbf{x}$  means  $x_1, \dots, x_n$  ( $n \geq 0$ ),  $\text{loc}$  means locations

Process  $P, Q, E, F ::=$

- 0
- ?g !h P
- loc:  $\mathbf{x}.\#_1 > u > \text{loc}' : \mathbf{y}.\#_r$ 
  - (loc:  $\mathbf{x}$ ) P
  - P || Q
  - loc:  $\mathbf{x} \# \mathbf{y}$
  - P & Q
  - loc >> P
  - P @ E @ F
- ; inaction
- ; globalized trigger, replicated
- ; globalized interaction: paired out|in, with only fusion
- ; continuations- reduces to loc:  $\#_1$  || loc:  $\#_r$  || loc: loc' :  $\mathbf{x} \# \mathbf{y}$
- ; visibility
- ; parallel composition
- ; explicit fusion
- ; globalized selection between alternatives
- ; projection of a process at a location
- ; choreography of: P normal, E exception, F finalizer

Guard  $g, h ::=$

- loc:  $u$  | loc:  $u \# v$  | g + g | g g | h + h | h h

# WS-CDL Formalisms (2)

- Linear Typing to support safety/liveness properties in the presence of non-determinism & non-termination [K. Honda, N. Yoshida]



# Choreography vs Orchestration (1)

- WS-CDL ➤ Global declaration of visible information but also specifying a residing location
  - BPEL ➤ Participant specific declaration of visible information
- WS-CDL ➤ Global Message Exchange (request and complementary matched accept) between 2 participants on a common Channel (w/ common reference and correlation)
  - BPEL ➤ Send a message to a participant, Receive a message from a participant, with participant specific viewpoints of messageInformation/partnerLocators/correlations
- WS-CDL ➤ Global Reactive Rules for declaratively prescribing normal/abnormal progress, based on the flexible information driven model
  - BPEL ➤ Participant specific viewpoints of control-flow (seq, flow, switch, controlLinks, exception/compensation handlers), based on the rigid imperative model

# Choreography vs Orchestration (2)

**WS-CDL** ➤ Common agreement of outcome (information alignment) across participants

**BPEL** ➤ No agreement of outcome across participants

**WS-CDL** ➤ Recursive Compositional model at the Choreography level and Templating allows reuse in different business contexts

**BPEL** ➤ Compositional model at the WSDL level

**WS-CDL** ➤ Generated computational logic of an individual collaborating participant can be BPEL, Java, C#, etc.

**BPEL** ➤ Computational logic of an individual participant is BPEL

# Collaboration Protocols

- Business protocols, such as Purchase Order Management
  - Involve creation of orders and change/cancel of in-flight orders
  - Found in
    - Financial protocols, such as the FIX, TWIST protocols
    - RosettaNet PIPs
- But can also be technical protocols, such as WS-Business Activity Protocol/WS-AtomicTransaction [Microsoft, IBM]
  - Define a classic/extended transaction model

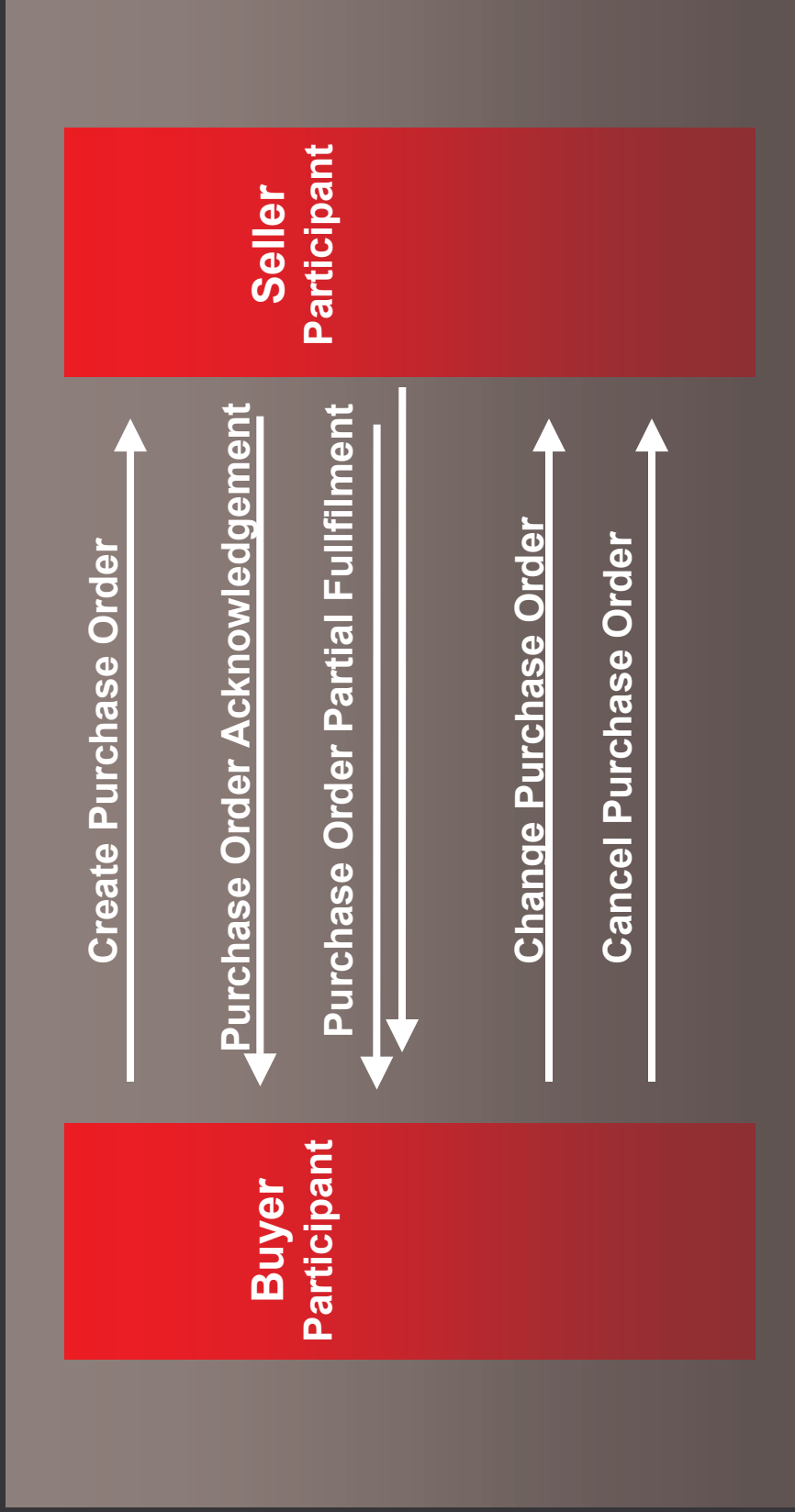
# Collaboration Protocol Requirements (1)

- Reusability in different contexts (industry, locale, etc.)
  - Allow specifying varying degrees of details incrementally
- Compositional model for enabling scalable modeling
  - Build smaller choreographies first
  - Combine them together to form a larger choreography

# Collaboration Protocol Requirements (2)

- *Global model*, across two or more collaborating participants, of:
  - Stuck-free Message Exchanges (requests matched with complementary accepts)
  - Channel-Passing, through 2 or more participants
  - State Creation/Changes, Race Conditions capturing
  - Common Reactive rules & Ordering Constraints (normal, exceptional), prescribing Information Driven computational progress in a declarative and flexible fashion
  - Information alignment, guaranteeing common understanding of exchanged information and state changes
  - Coordinated Progress (normal, exceptional)
- With expressiveness greater than what CCS model offers
- And guarantees of safety/liveness properties in the presence of non-determinism & non-termination

# Real-life Business Collaboration Protocol: Order Management

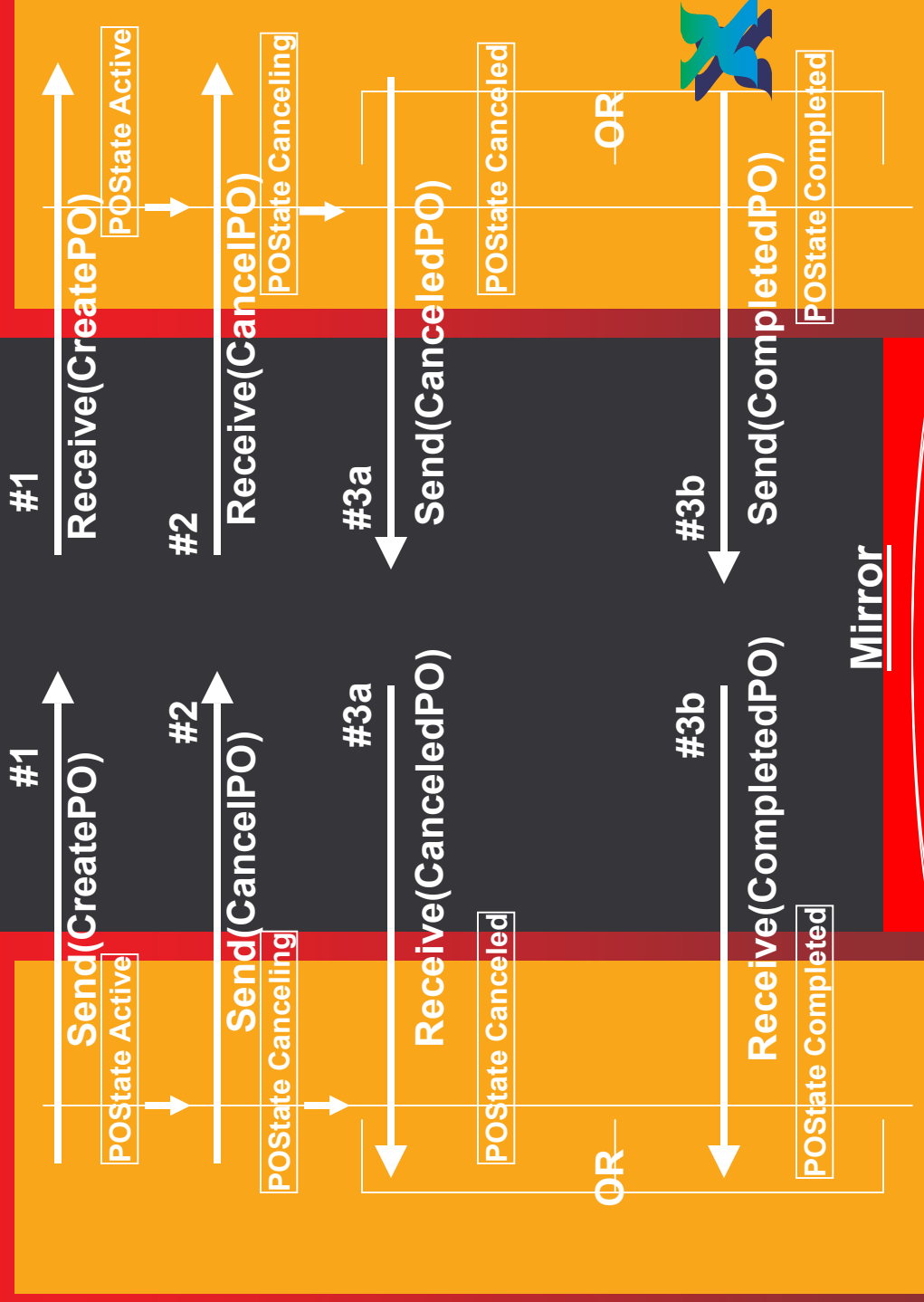


# Modeling a Collaboration Protocol with BPEL

Design individual puzzle pieces first and put them together later

- **Step 1:** Design one piece
  - Model the localized behavior of one participant: Buyer
- **Step 2:** Design another piece
  - Model the localized behavior of another participant: Seller, Shipper, etc.
- **Oops!** The individually designed pieces do not fit together
  - The composed behavior of the individually modeled behaviors may capture an un-desired global system behavior (see #3b, Seller participant in the following slide)

# Modeling a Collaboration Protocol with BPEL: Undesired Global Behavior



Buyer-Participant "A"

Step-1

Step-2

Seller-Participant "B"

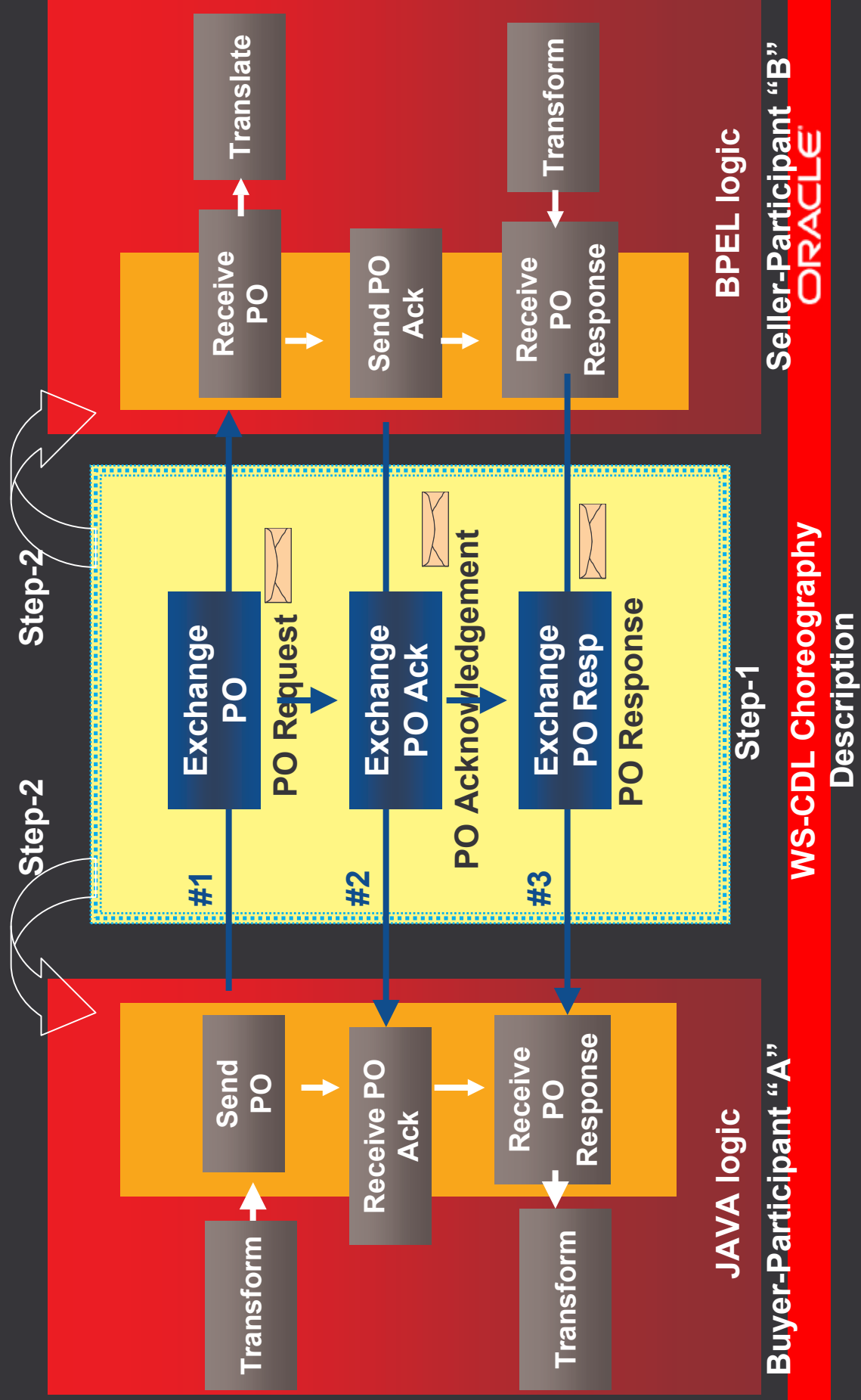


# Modeling a Collaboration Protocol with WS-CDL

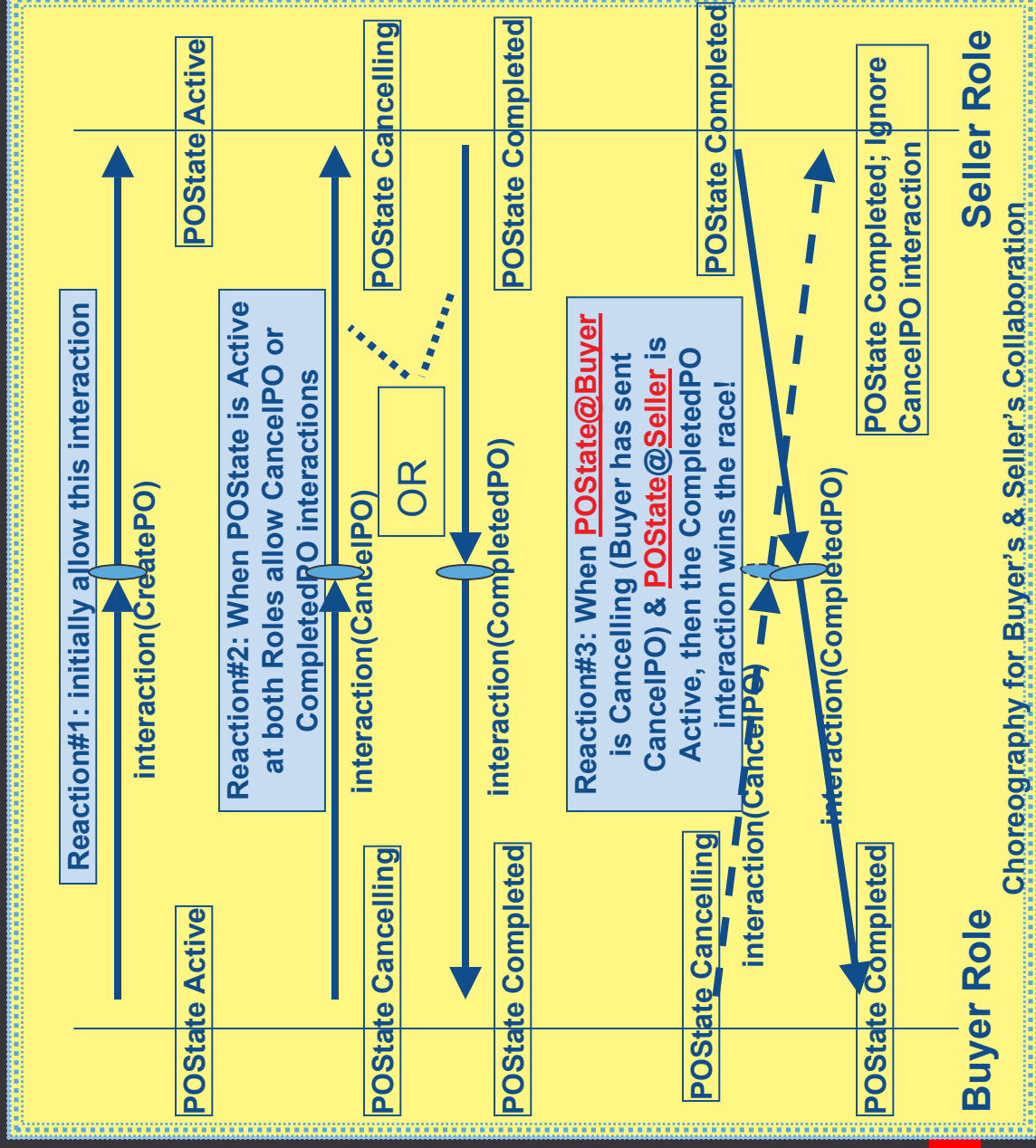
“Big-Picture” first and cut puzzle pieces later

- **Step 1:** Look one picture
  - Start by modeling the collaboration between the Buyer participant and the Seller participant
- **Step 2:** Then cut the smaller pieces
  - From WS-CDL we can generate BPEL/Java/C# logic for each participant, ie.
    - Java logic for the Buyer
    - BPEL logic for the Seller

# Modeling a Collaboration Protocol with WS-CDL



# Modeling a Collaboration Protocol with WS-CDL: *Desired Global Behavior*



Choreography for Buyer's & Seller's Collaboration

# WS-CDL in Detail

ORACLE

# WS-CDL Concepts

- Typing
  - Information Type
  - Token, Token Locator
- Identifying & Coupling Collaborating Participants
  - Role
  - Relationship
  - Participant
- Information Driven Collaborations
  - Channel Type
  - Variable Definitions
  - Activities
    - interaction, perform, assign, noaction
  - WorkUnit/Reaction
  - Choreography
  - Package, Import

# Typing

- Information type
  - Aliases WSDL types, XSD type/element
  - Supports other type systems
- Token type
  - Specify name and type as an alias to a piece of information within a document
- Token Locator type
  - Specify rules for selecting a piece of information within a document

# Typing Syntax

```
<informationType name="ncname"  
  type="qname"? |  
  element="qname"? />  
  
<token name="ncname" informationType="qname" />  
  
<tokenLocator tokenName="qname"  
  informationType="qname"  
  query="XPath-expression"? />
```

# Roles, Relationships, Participants

- Role type
  - Enumerate the observable behavior, specified by one or more Behavior type(s), a collaborating participant exhibits
  - Behavior type specifies the operations supported
    - Optional WSDL interface type
- Relationship type
  - Specify the mutual commitments, in terms of the Roles/ Behavior types, two collaborating participants are required to provide
- Participant type
  - Enumerate a set of one or more Roles that a collaborating participant plays



# Roles, Relationships, Participants Syntax

```
<role name="ncname" >  
  <behavior name="ncname"  
    interface="qname"? />+  
</role>  
  
<relationship name="ncname">  
  <role type="qname" behavior="ncname" />  
  <role type="qname" behavior="ncname" />  
</relationship>  
  
<participant name="ncname">  
  <role type="qname" />+  
</participant>
```

# Channel (1)

- Realizes a *dynamic* point of collaboration, through which collaborating participants interact
  - Where to & how to communicate a message
    - Specify the *Role/Behavior* and the *Reference* of a collaborating participant
    - Identify an *Instance* of a Role
  - Identify an instance of a conversation between two or more collaborating participants
    - A conversation groups a set of related message exchanges

# Channel (2)

- One or more channel(s) MAY be passed around from a Role to one or more other Role(s), possibly in a daisy fashion through one or more intermediate Role(s), creating new points of collaboration dynamically
  - A Channel type MAY restrict the types of Channel(s) allowed to be exchanged between the Web Services participants, through this Channel
  - A Channel type MAY restrict its usage, by specifying the number of times a Channel can be used

# Channel Syntax

```
<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 type="qname" />+  
</reference>  
  
<identity>  
  <token type="qname" />+  
</identity>*  
</channelType>
```

# Variables (1)

Capture instance information about objects in a collaboration

- Affect the progress of collaborations between collaborating participants
- Variable types:
  - Information Exchange Variables: define instances of exchanged docs between Roles in an interaction
  - State Variables: define instances of state information at a Role
  - Channel Variables: define instances of channel types
- And their definitions:
  - Specify the type of value a variable contains using `informationType`, `channelType`
  - Specify the Role of the collaborating participant a variable resides in

# Variables (2)

- The value of a variable
  - Is available to all the Roles by initializing them prior to the start of a Choreography. Common variables contain information that is common knowledge to two or more Roles
  - Can be made available at a Role by populating them as a result of an interaction
  - Can be made available at a Role by assigning data from other information. Locally defined variables that contain information created and changed locally by a Role. They can be Information Exchange, State or Channel Variables
  - Can be used to determine the decisions and actions to be taken within a Choreography

# Variable Syntax

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

# Activities: Interaction (1)

Enable collaborating participants to communicate and align information

- Message exchange(s) between two Roles within a Relationship
  - Request & Accept of an operation through a common channel
    - one way or request-response operation
    - information flow
      - request direction: fromRole towards toRole
      - response direction: toRole towards fromRole



# Activities: Interaction (2)

- State recordings at a Role
  - Create new, modify existing variables at a Role
- Information Alignment: agreement on their common understanding of the
  - State changes of variables that reside in one Role with the state changes of variables that reside in the other Role
  - Values of the exchanged messages from one Role to the other Role

# Interaction Syntax

```
<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 messageContentType="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>
```

# Activities: Assign

- State recordings at a Role
  - Create new, modify existing variables at a Role

```
<assign role="qname">  
  <copy name="ncname" >  
    <source variable="XPath-expression" />  
    <target variable="XPath-expression" />  
  </copy>+  
</assign>
```

# WorkUnit/Reaction

- Information driven model, where a Reaction rule guards a set of activities, by prescribing the constraints on information that need, to be satisfied for making normal/abnormal progress
  - Reaction Guard expresses interest on the availability of one or more variable information
  - When the variable is/becomes available and the guard condition evaluates to true, the enclosed activities are enabled
  - Guard condition may be on variables that reside on different Roles
- Repeat: marks the re-enablement of a workunit
- isAligned(var1, var2, rel) specifies interest on the alignment of variables in two Roles within a Relationship

# WorkUnit/Reaction Syntax

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

# Enablement within a set of Activities

- **Sequential:** specifies the enablement of activities in a sequential fashion
- **Parallel:** specifies the enablement of activities in a parallel fashion
- **Choice:** specifies the enablement of one activity from a set of two or more activities

```
<sequence>  
  Activity+  
</sequence>
```

```
<parallel>  
  Activity+  
</parallel>
```

```
<choice>  
  Activity+  
</choice>
```

# Choreography

- Combines actions with common behavioral characteristics, forming a collaboration unit of work
  - Enumerate all the binary relationships interactions act in
  - Localize the visibility of variables
    - Using variable definitions
  - Prescribe alternative patterns of behavior
    - Using workunits/reactions
  - Enable Recovery
    - Using workunits/reactions
    - Backward: handle exceptional conditions
    - Forward: finalize already completed activities

# Choreography Syntax

```
<choreography name="ncname"
  complete="xsd:boolean XPath-expression"?
  isolation="dirty-write" |
  "dirty-read" | "serializable"?
  root="true" | "false"? >

  <relationship type="qname" />+
  variableDefinitions?
  Choreography*

  Activity

  <exception name="ncname">
    WorkUnit+
  </exception?>

  <finalizer name="ncname">
    WorkUnit
  </finalizer>
</choreography>
```



# Choreography Composition (1)

- WS-CDL's compositional model allows scalable modeling
  - Build smaller choreographies first
  - Combine them together to form a larger choreography
  - Then do participant projections to get a participant's viewpoint
    - Composition makes participant projections easier
    - Without the big-picture of a “business process” in perspective, participant projections do not make sense

# Choreography Composition (2)

- Choreographies can be recursively combined to form new Choreographies, using the perform activity

```
<perform choreographyName="qname">  
  <alias name="ncname">  
    <this variable="XPath-expression" role="qname" />  
    <free variable="XPath-expression" role="qname" />  
  </alias>+  
</perform>
```

# Package, Import, Templates

## Facilitate modularity, reusability

- A Choreography package allows combining choreography types under a common namespace:
  - informationType, token, tokenLocator
  - role, relationship, participant
  - channelType
  - variableDefinitions
  - choreography
- An import allows using choreography types defined in another Choreography package
- Templating allows specifying varying degrees of details incrementally, enabling reusability of choreographies in different contexts (industry, locale, etc.)

# Package, Import Syntax

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

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

# Summary

ORACLE

# Oracle: Orchestration and Choreography Strategy

- Orchestration
  - Oracle commitment to BPEL within middle tier runtime, management and development tooling
  - Direct product impacts: OracleAS Integration and Oracle Jdeveloper
- Choreography
  - W3C WS-CDL specification Editor and key participant in W3C Choreography Working Group
  - Working with and recruiting early adopters to drive specific product requirements

# Conclusions

- Choreography complements Orchestration
- Choreography is concerned with global, multi-party, peer-to-peer collaborations, between application components, distributed within or across an organizations trusted domain
- WS-CDL is based on a formal model
- WS-CDL is currently under standardization in W3C Choreography WG
  - Please download the latest working draft and comment