Adobe®

Web Services Choreography

Charlton Barreto

Senior Computer Scientist

2005-May-11

Agenda

- Overview
- Components
- Motivations
- Using CDL
- Comparison
- Approach
- Example
- Summary

Overview

- **What is Choreography?**
  - Peer-to-peer global model operating between loosely coupled components and systems that operate to solve a common task
  - Each component or system may reside on different networks and have different levels of reliability and performance
  - Establishes a global behavioural contract
  - Used across domains of control to ensure harmony (interoperability, etc.)

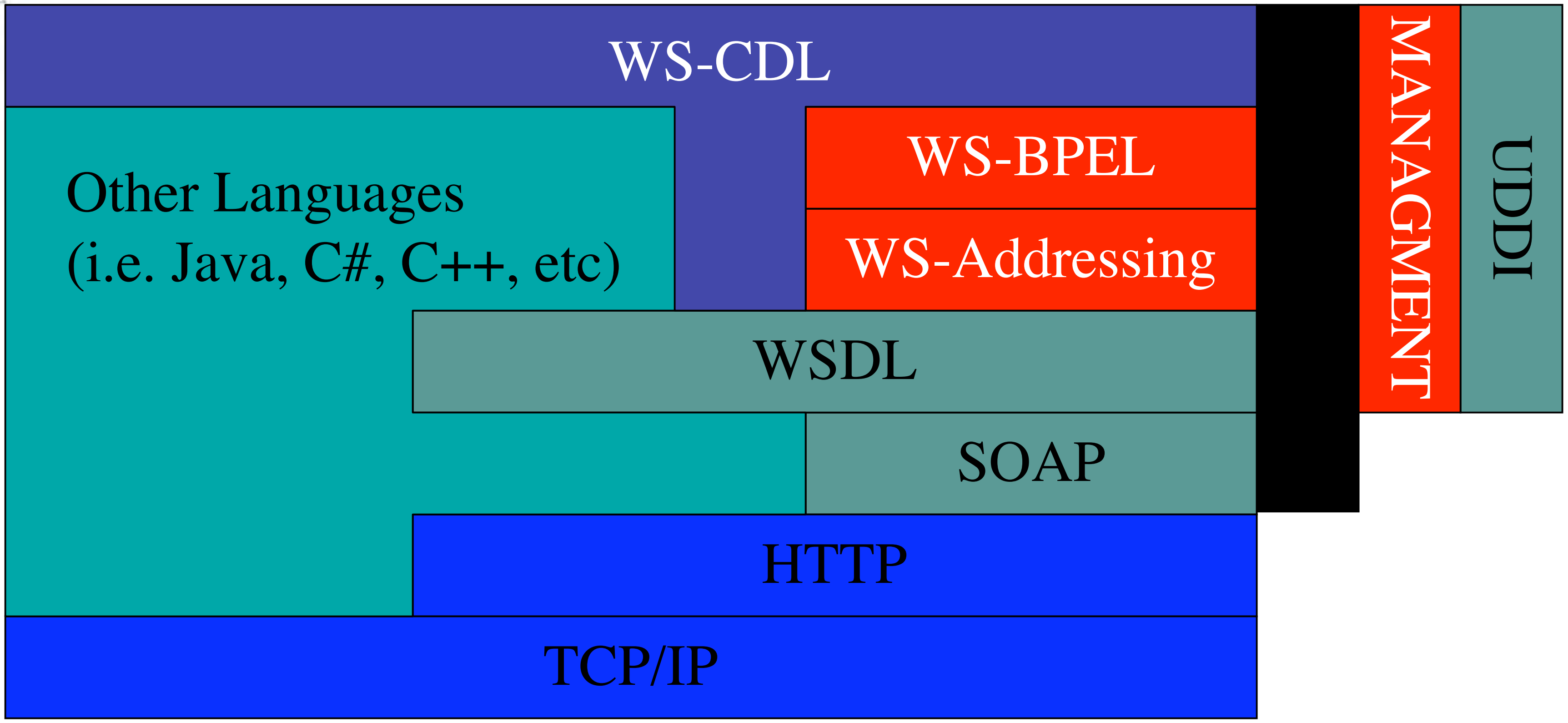
Overview

- **What is WS-Choreography?**
  - A working group in W3C tasked with defining a language for describing peer-to-peer service interactions from a neutral perspective
  - Based on a formalized description of external observable behaviour across domains
  - Current status
    - Requirements document (published March 2004)
    - Model Overview document (published April 2004)
    - 1st Working Draft of the WS-CDL specification (published April 2004)
    - Last Call Working Draft published Dec 2004
    - Last Call period has just ended

Overview

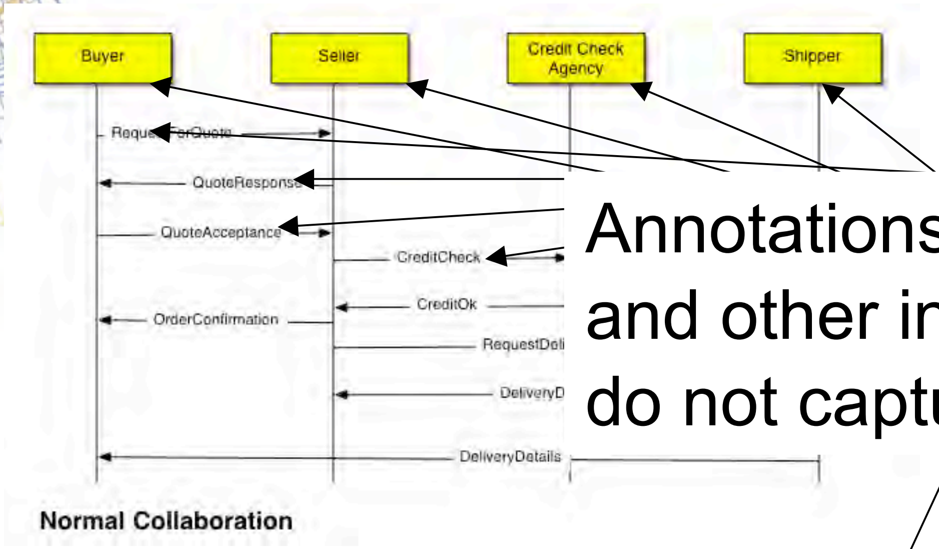
- **What is CDL?**
  - CDL is the Choreography Description Language (WS-CDL)
  - CDL can be used to describe collaboration protocols of cooperating [Web] Service participants in which:
    - Services act as peers
    - Interactions may be long-lived and stateful
  - A CDL description is a multi-participant contract that describes - from a neutral or global viewpoint - the common observable behaviour of the collaborating participants

Overview



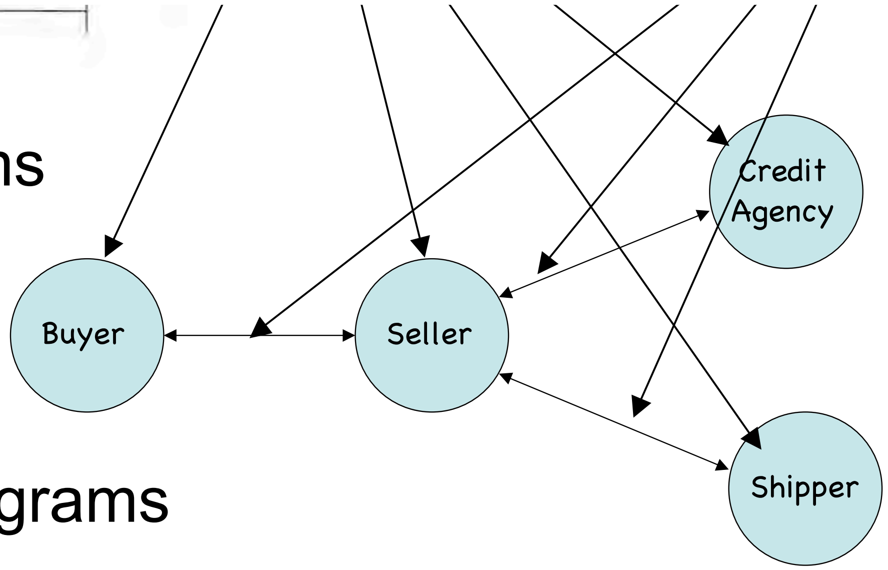
Legacy
Available
Nascent
Missing

Motivations



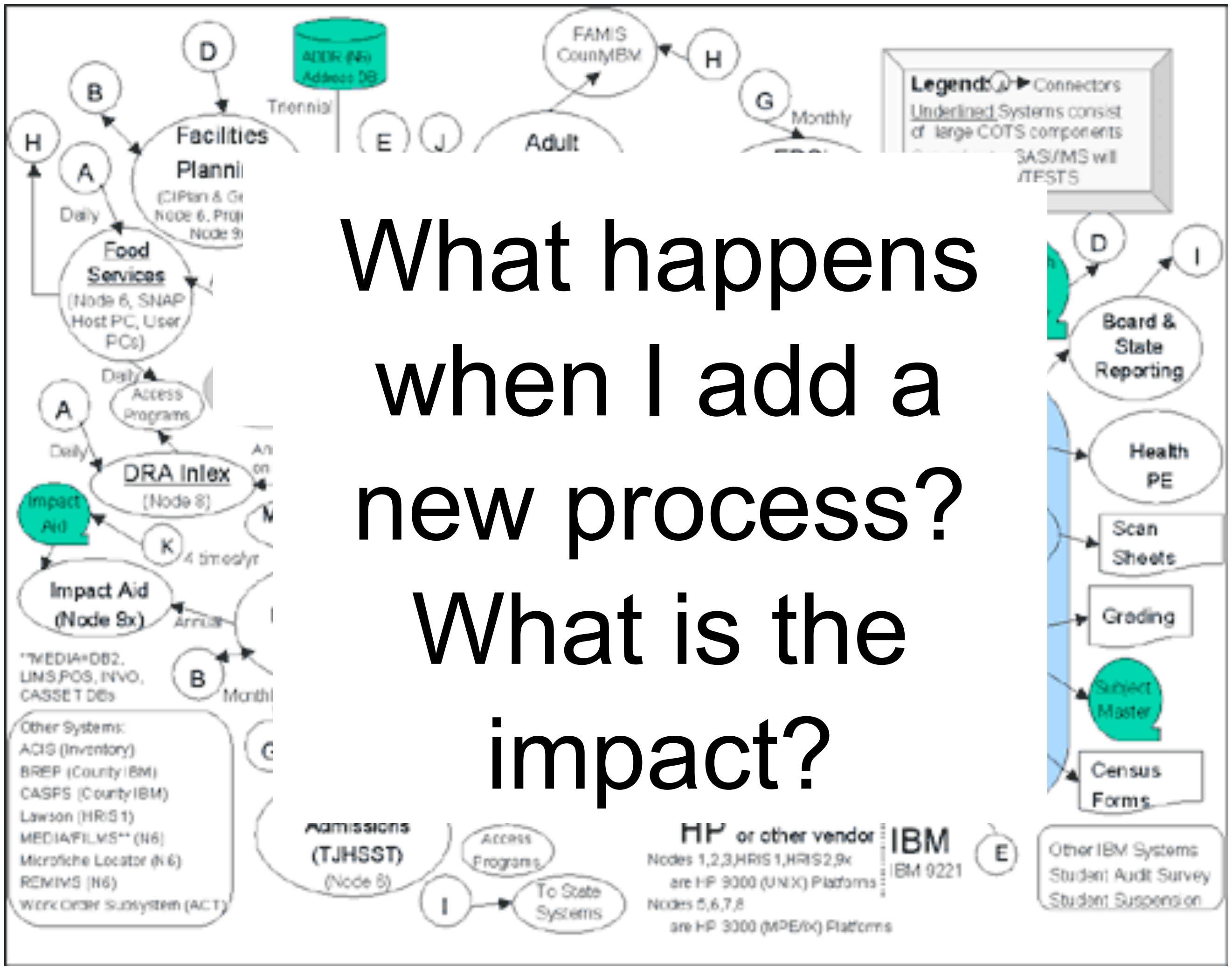
Annotations are used to record timings and other information that these models do not capture

UML Sequence diagrams



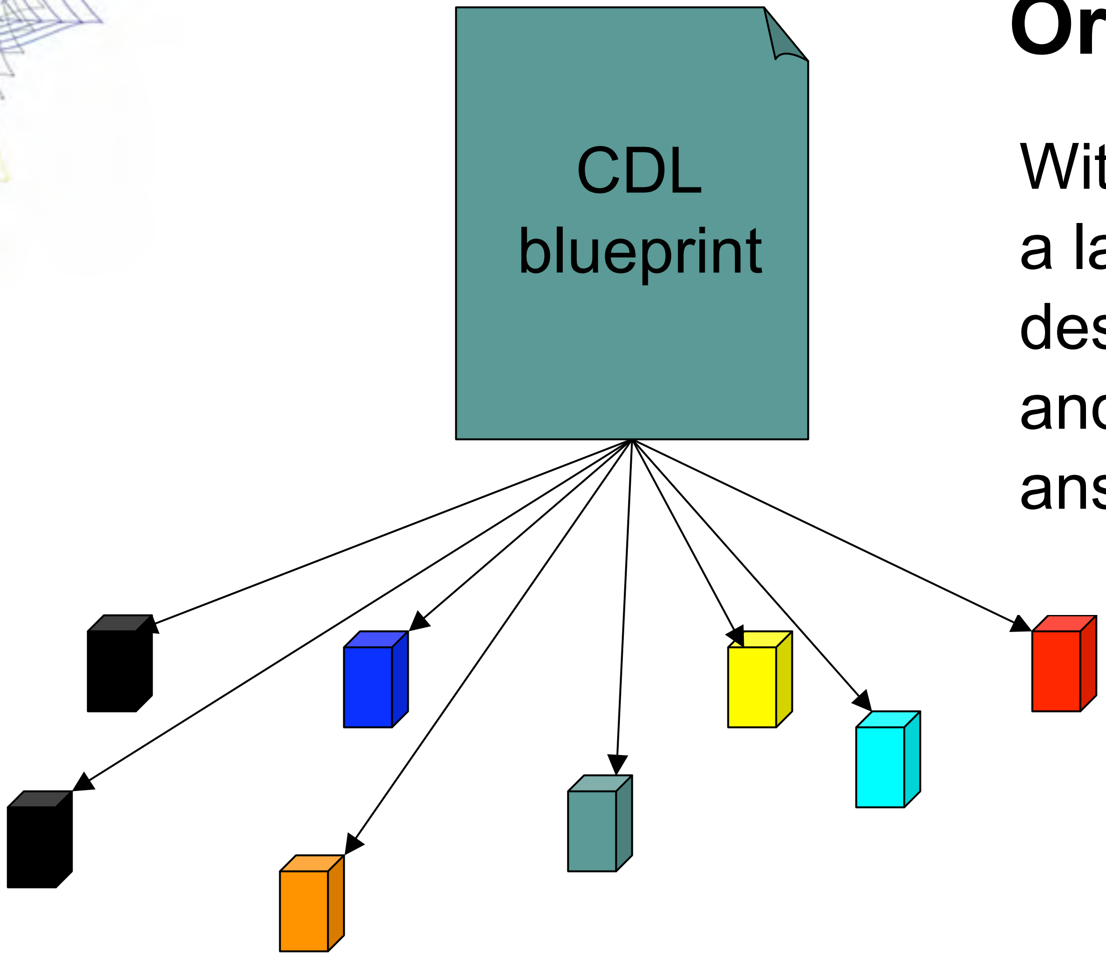
Bubble and stick diagrams

Motivations



What happens when I add a new process? What is the impact?

Motivations



Order from chaos

With CDL we now have a language that properly describes the blueprint and allows you to answer these questions

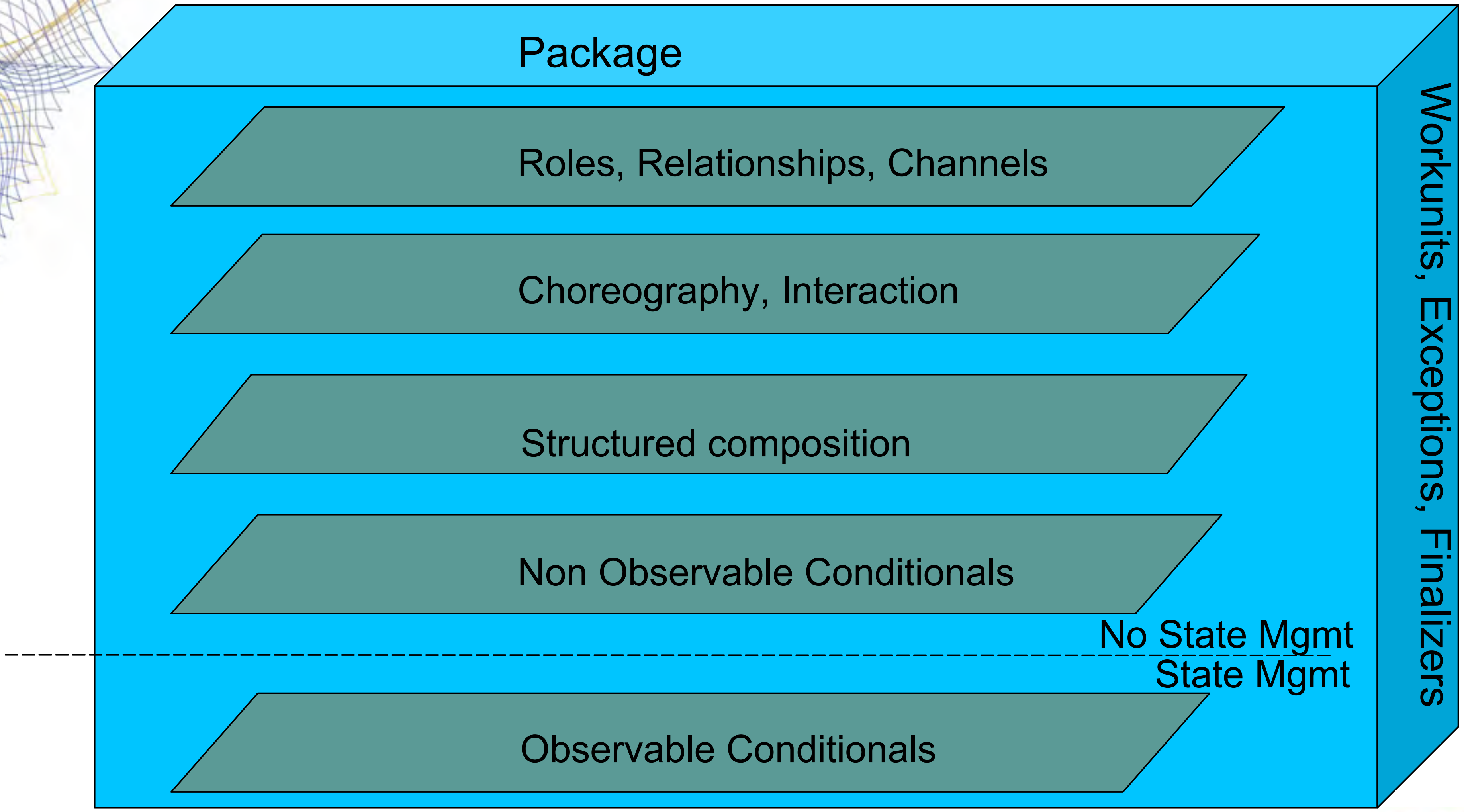
Using CDL

- **One would use CDL**
  - To ensure effective interoperability of Services - guaranteed as Services must conform to a common behavioural multi-party contract specified in the CDL
  - To create more robust Services - they can be validated statically and at runtime against a CDL
  - To reduce the cost of implementing Services by ensuring conformance to expected behaviour described in the CDL
  - To formally encode agreed multi-party business protocols such as fpML, FIX, SWIFT and TWIST

Using CDL

- **WS-CDL is a design language**
  - For describing the technical contract that need to be enforced between peer participants
  - That can be used to generate end point skeletal behaviour (i.e. only the observable business logic)
  - That can be used to generate end point monitors that can enforce the contractual behaviour of the end point
  - That can be used to generate test scripts or used in conjunction with test scripts to ensure that the contract is valid.

What does CDL look like?


Package
Roles, Relationships, Channels
Choreography, Interaction
Structured composition
Non Observable Conditionals
No State Mgmt
State Mgmt
Observable Conditionals
Workunits, Exceptions, Finalizers

Components

- **Package**
  - Information - data
  - Participants - service locations that implement a set of roles
  - Roles - service end points (WSDL)
  - Relationships - association between roles
  - Channel types – communication channel definitions
  - Choreography - the multi-party behavioural contract details

Components

- **Choreography**
  - Composable unit that describes the behaviour between the defined participants based on the roles that they play and the channels that they use to interact
- **Channel instance**
  - The communication 'pipe' established between two roles - channel type, associated with the instance, may permit a one instance to transfer others as valid messages (i.e. channel passing)
- **Interaction**
  - Describes communication between two participants along a channel instance (e.g. request/response, one-way, etc.)

Components

- **Activities**
  - Sequence - describes a sequence of activities
  - Parallel - describes a parallel set of activities
  - Choice - describes a mutually exclusive set of activities
  - Perform - describes an enclosed choreography to be performed
  - Variable assignment - described the assignment of variable information

Components

- **Workunits**
  - A grouping construct for a set of contained activities
  - Can define a guard condition, to make these activities conditional
  - Can express a repetition condition, to indicate whether the activities should be repeated
  - Can be used for synchronization based the variables used in the guard condition - if variables used in the condition are not currently set, then the evaluation of the condition would suspend until all the information was available.

Components

- **Exceptions**
  - A special kind of workunit associated with a choreography that is enabled when an exception is detected
- **Finalizers**
  - Defines a set of activities to be performed at choreography completion
    - When a 'performed' choreography completes successfully, it can enable one or more finalizers
    - One of these finalizers can then be selected by the performing choreography to complete the work associated with the 'performed' choreography (e.g. Confirm or Cancel)
  - Can be used to provide a typical compensation model, as well as support a range of coordination models

Components

```
<choreography  name="priceRequest" root="false">
        <relationship type="tns:BuyerSeller"/>
        <variableDefinitions>
                <variable name="SellerA" informationType="tns:channelType"/>
                <variable name="SellerB" informationType="tns:channelType"/>
        <\variableDefinitions>
        <parallel>
                <!—Two interactions with two variables  … -- >
                <interaction channelVariable="tns:SellerA" operation="priceRequest" align="false" initiateChoreography="true">
                        <participate relationship="tns:BuyerSeller" fromRole="tns:Buyer" toRole="tns:Seller"/>
                        <exchange messageContentType="tns:priceReqMessage" action="request" />
                        <exchange messageContentType="priceRespMessage" action="respond" />
                        <!—Record the price at the buyer role for later consolidation into "prices" -- >
                        <record role="tns:Buyer" action="response">
                                <source variable="cdl:getVariable(tns:price, PR/????, tns:Seller)"/>
                                <target variable="cdl:getVariable(tns:priceA, tns:Buyer)"/>
                        </record>
                </interaction>

                <interaction channelVariable="tns:SellerB" operation="priceRequest" align="false" initiateChoreography="true">
                        <participate relationship="tns:BuyerSeller" fromRole="tns:Buyer" toRole="tns:Seller"/>
                        <exchange messageContentType="tns:priceReqMessage" action="request" />
                        <exchange messageContentType="priceRespMessage" action="respond" />
                        <!—Record the price at the buyer role for later consolidation into "prices" -- >
                        <record role="tns:Buyer" action="response">
                                <source variable="cdl:getVariable(tns:price, PR/????, tns:Seller)"/>
                                <target variable="cdl:getVariable(tns:priceB, tns:Buyer)"/>
                        </record>
                </interaction>
        <\parallel>
<\choreography>
```

Interaction Dependencies

- **To define a choreography of interactions, one must first**
  1) define roleTypes
  2) define relationshipTypes
  3) define informationTypes
  4) define tokenType
  5) define channelTypes

Role Types

```
<roleType name="BuyerRoleType">
  <description type="documentation">The Behavior embodied by a buyer</description>
  <behavior name="BuyerBehavior" />
</roleType>

<roleType name="SellerRoleType">
  <description type="documentation">The behavior embodied by a seller</description>
  <behavior name="SellerBehavior" />
</roleType>

<roleType name="CreditCheckerRoleType">
  <description type="documentation">The behavior embodied by a credit checker </description>
  <behavior name="CreditCheckerBehavior" />
</roleType>

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

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

Relationship Types

```
<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>
```

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

Information Types

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

Token Types

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

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

Channel Types

```
<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>
```

```
<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> >
```

Channel Types

```
<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 Dependencies

- **Then, to complete defining a choreography, one must**
  1) declare variables
  2) declare relationshipTypes

Choreography

```
<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
withoulinkage
      </description>
    </variable>
    <variable name="barteringDone"  informationType="BooleanType"  roleTypes="BuyerRoleType SellerRoleType">
      <description type="description">Has Bartering Finished flag</description>
    </variable>
  </variableDefinitions>
```

Interactions & Ordering

```
<?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>

…………

 <choreography name="Main" root="true">
   <description type="description">Collaboration between buyer, seller, shipper, credit chk</description>

      …………

    <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>

        …………

    </sequence>
 </choreography>
</package>
```

Bartering Process

```
    <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>

     …………
```

Bartering Process

```
    …………
        <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>
```

Dependent WorkUnits

```
<parallel>
   <workunit name="Repeat until bartering has been completed" repeat="barteringDone = false">
      ……
   </workunit>

   <workunit name="Process Order" guard="barteringDone = true" blocking="true">
      ……
   </workunit>
</parallel>
```

Dependent WorkUnits

```
<parallel>
   <workunit name="Check Credit Rating">
      <sequence>
         <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>
            <sequence>
               <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>
               <assign roleType="SellerRoleType">
                   <copy name="copy">
                      <source expression="false" />
                      <target variable="cdl:getVariable('creditRatingOk','','')" />
                   </copy>
               </assign>
            </sequence>

    ...........
```

Dependent WorkUnits

```
  …………

          <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>
             <assign roleType="SellerRoleType">
                <copy name="copy">
                   <source expression="true" />
                   <target variable="cdl:getVariable('creditRatingOk','','')" />
                </copy>
             </assign>
          </sequence>
       </choice>
    </sequence>
 </workunit>


 <workunit name="Request Delivery" guard="creditRatingOk = true" blocking="true">
    ……
 </workunit>
</parallel>
```

WorkUnits

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) {
                Body
        }
```

WorkUnits

Non-blocking

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

A typical order of evaluation is as follows:

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

Which equates to (in pseudo code):

```
while (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:

```
if (G) {
    Body
}
```

Comparison

- **BPEL**
  - Orchestration implies a centralized control mechanism
  - Recursive Web Service Composition
  - Executable language
  - Requires Web Services
- **CDL**
  - Choreography has no centralized control
    - Control is shared between domains
  - Description language
  - Does not require Web Services,
    - Targeted to deliver over them
  - CDL doesn't see BPEL as unique or different to any other end-point language
    - One can generate to BPEL just as easily as to Java

Approach

- **Based on simple contract-like mechanisms**
  - Deadlock-freedom (Kobayashi, 99, 00)
  - Liveness (Kobayashi, 01; Yoshida, et al, 02)
  - Security (Abadi et al; Cardelli and Gordon; Berger, Honda, Yoshida)
  - Resource management (Tofte; Kobayashi; Gordon and Dal Zillio; Yoshida, et al)
  - Race-condition detection (refs)
  - Extensions to CCS/CSP and π-calculus (Milner)

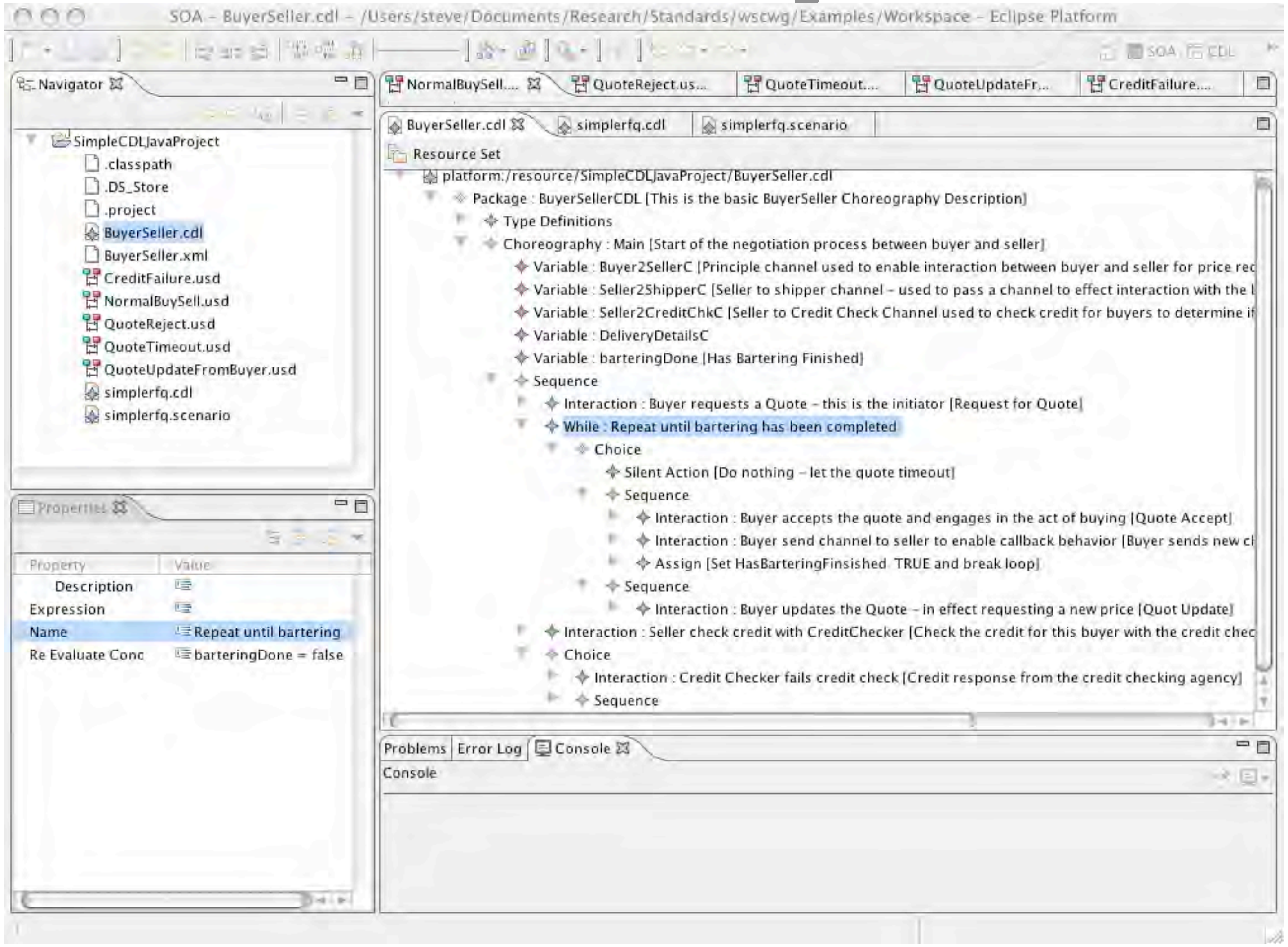
Approach

Model	Completeness	Compositionality	Parallelism	Resources
Turing Machines	☑	☒	☒	☑
Lambda	☑	☑	☒	☒
Petri Nets	☑	☒	☑	☑
CCS	☑	☑	☑	☒
π	☑	☑	☑	☑

Example

- **Tools**
  - Bartering process (but without variables)
    - Demo WS-CDL editor
    - Tree based editor based on structural clarity (see workunit explanation)
    - Demo testing
    - Testing a choreography by simulating messages that make up interactions.
    - Correct set of messages
    - Incorrect set of messages - results in a “SEVERE” error warning

WS-CDL Editor Buyer/Seller



SOA – BuyerSeller.cdl – /Users/steve/Documents/Research/Standards/wscwg/Examples/Workspace – Eclipse Platform

Navigator

```
SimpleCDLJavaProject
    .classpath
    .DS_Store
    .project
    BuyerSeller.cdl
    BuyerSeller.xml
    CreditFailure.usd
    NormalBuySell.usd
    QuoteReject.usd
    QuoteTimeout.usd
    QuoteUpdateFromBuyer.usd
    simplerfq.cdl
    simplerfq.scenario
```

Properties

Property	Value
Description	
Expression	
Name	Repeat until bartering
Re Evaluate Conc	barteringDone = false

NormalBuySell.... | QuoteReject.us... | QuoteTimeout.... | QuoteUpdateFr... | CreditFailure....

BuyerSeller.cdl | simplerfq.cdl | simplerfq.scenario

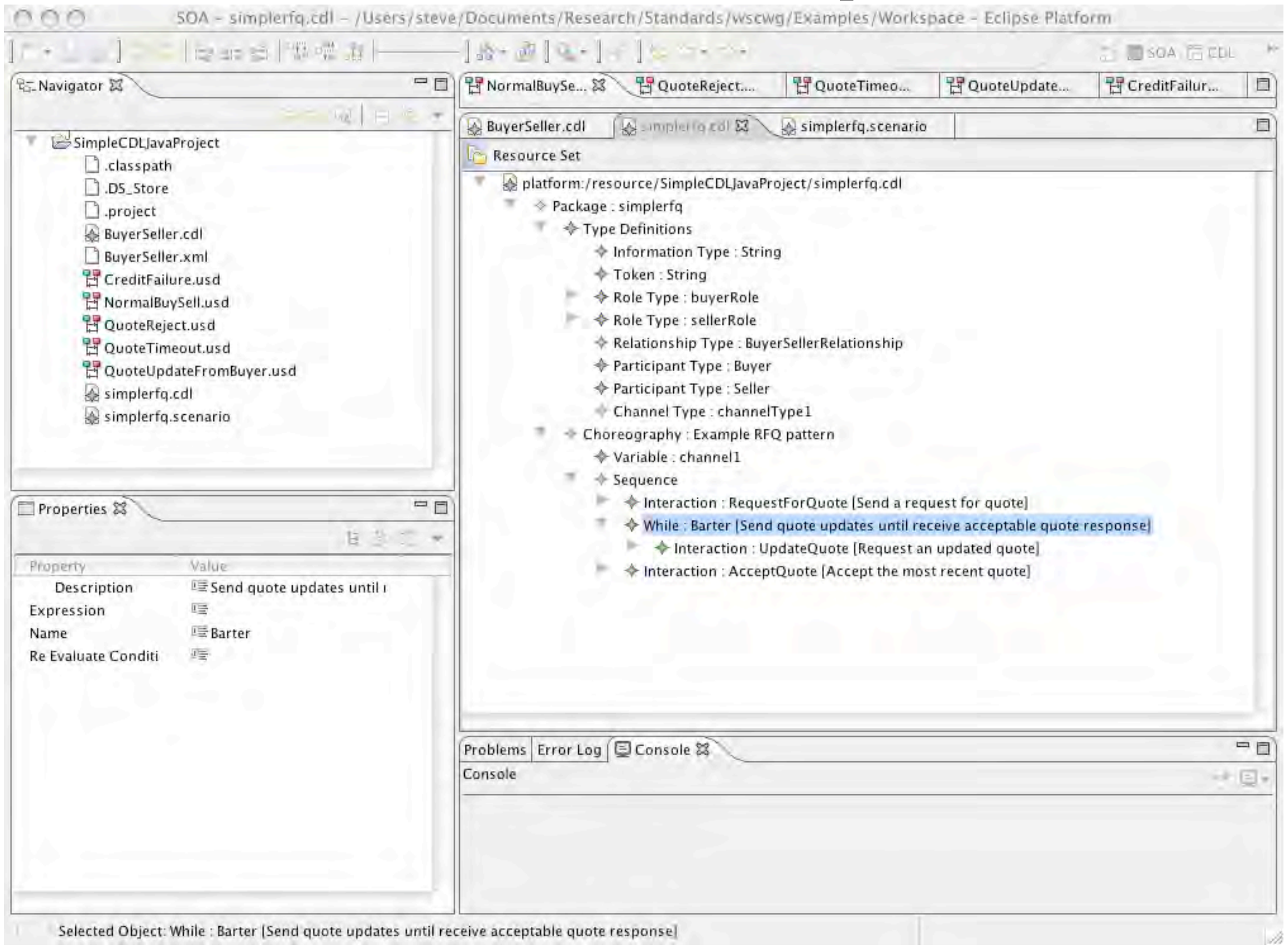
Resource Set

```
platform:/resource/SimpleCDLJavaProject/BuyerSeller.cdl
  Package : BuyerSellerCDL [This is the basic BuyerSeller Choreography Description]
    Type Definitions
    Choreography : Main [Start of the negotiation process between buyer and seller]
      Variable : Buyer2SellerC [Principle channel used to enable interaction between buyer and seller for price rec
      Variable : Seller2ShipperC [Seller to shipper channel - used to pass a channel to effect interaction with the l
      Variable : Seller2CreditChkC [Seller to Credit Check Channel used to check credit for buyers to determine if
      Variable : DeliveryDetailsC
      Variable : barteringDone [Has Bartering Finished]
      Sequence
        Interaction : Buyer requests a Quote - this is the initiator [Request for Quote]
        While : Repeat until bartering has been completed
          Choice
            Silent Action [Do nothing - let the quote timeout]
            Sequence
              Interaction : Buyer accepts the quote and engages in the act of buying [Quote Accept]
              Interaction : Buyer send channel to seller to enable callback behavior [Buyer sends new ch
              Assign [Set HasBarteringFinsished TRUE and break loop]
            Sequence
              Interaction : Buyer updates the Quote - in effect requesting a new price [Quot Update]
        Interaction : Seller check credit with CreditChecker [Check the credit for this buyer with the credit chec
        Choice
          Interaction : Credit Checker fails credit check [Credit response from the credit checking agency]
          Sequence
```

Problems | Error Log | Console

Console

WS-CDL Editor Simple RFQ



SOA – simplerfq.cdl – /Users/steve/Documents/Research/Standards/wscwg/Examples/Workspace – Eclipse Platform

Navigator

```
SimpleCDLJavaProject
    .classpath
    .DS_Store
    .project
    BuyerSeller.cdl
    BuyerSeller.xml
    CreditFailure.usd
    NormalBuySell.usd
    QuoteReject.usd
    QuoteTimeout.usd
    QuoteUpdateFromBuyer.usd
    simplerfq.cdl
    simplerfq.scenario
```

Properties

Property	Value
Description	Send quote updates until r
Expression	
Name	Barter
Re Evaluate Conditi	

NormalBuySe... | QuoteReject... | QuoteTimeo... | QuoteUpdate... | CreditFailur...

BuyerSeller.cdl | simplerfq.cdl | simplerfq.scenario

Resource Set

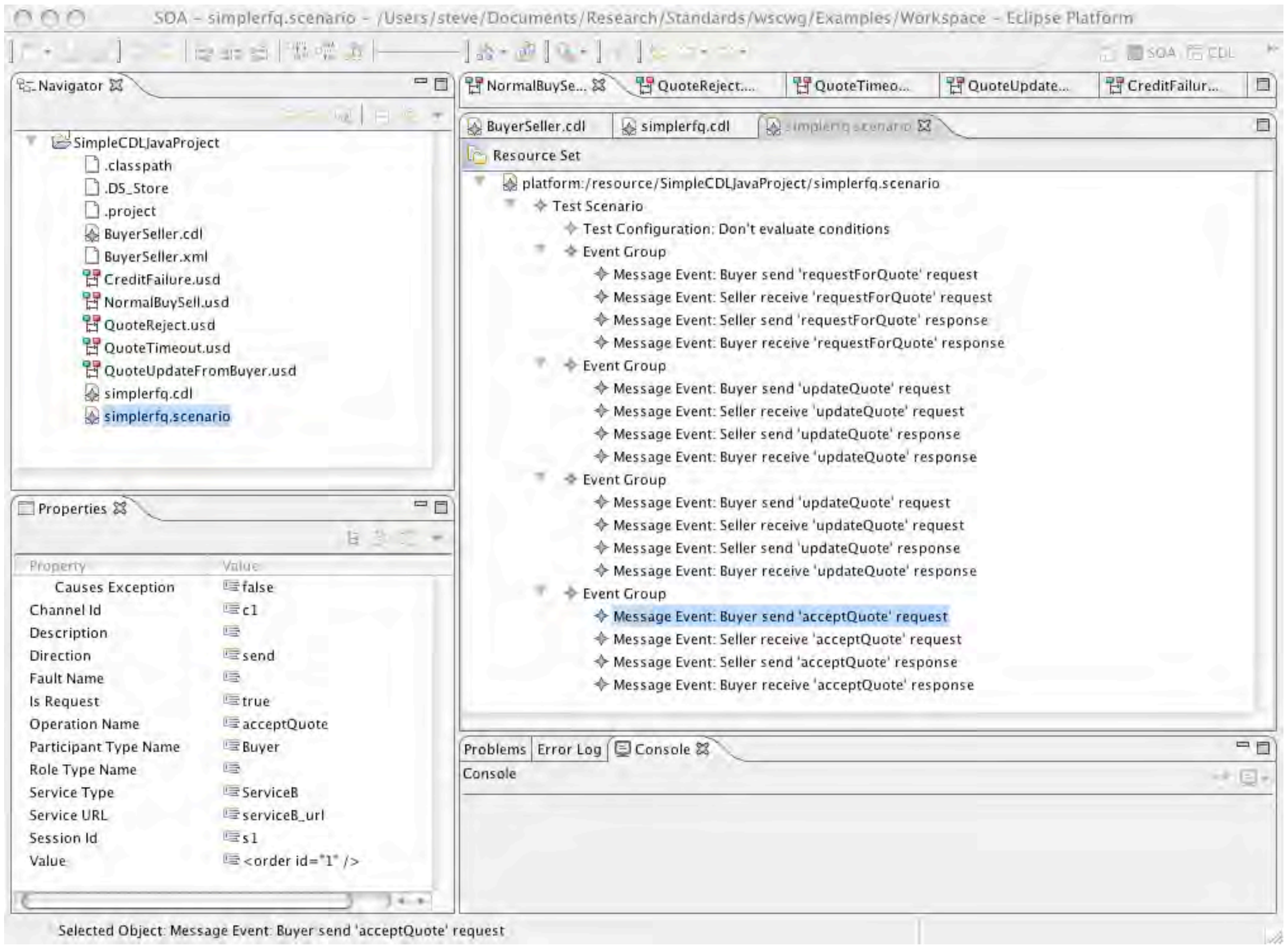
```
platform:/resource/SimpleCDLJavaProject/simplerfq.cdl
    Package : simplerfq
        Type Definitions
            Information Type : String
            Token : String
            Role Type : buyerRole
            Role Type : sellerRole
            Relationship Type : BuyerSellerRelationship
            Participant Type : Buyer
            Participant Type : Seller
            Channel Type : channelType1
        Choreography : Example RFQ pattern
            Variable : channel1
            Sequence
                Interaction : RequestForQuote [Send a request for quote]
                While : Barter [Send quote updates until receive acceptable quote response]
                    Interaction : UpdateQuote [Request an updated quote]
                Interaction : AcceptQuote [Accept the most recent quote]
```

Problems | Error Log | Console

Console

Selected Object: While : Barter [Send quote updates until receive acceptable quote response]

WS-CDL Good Test Script



SOA – simplerfq.scenario – /Users/steve/Documents/Research/Standards/wscwg/Examples/Workspace – Eclipse Platform

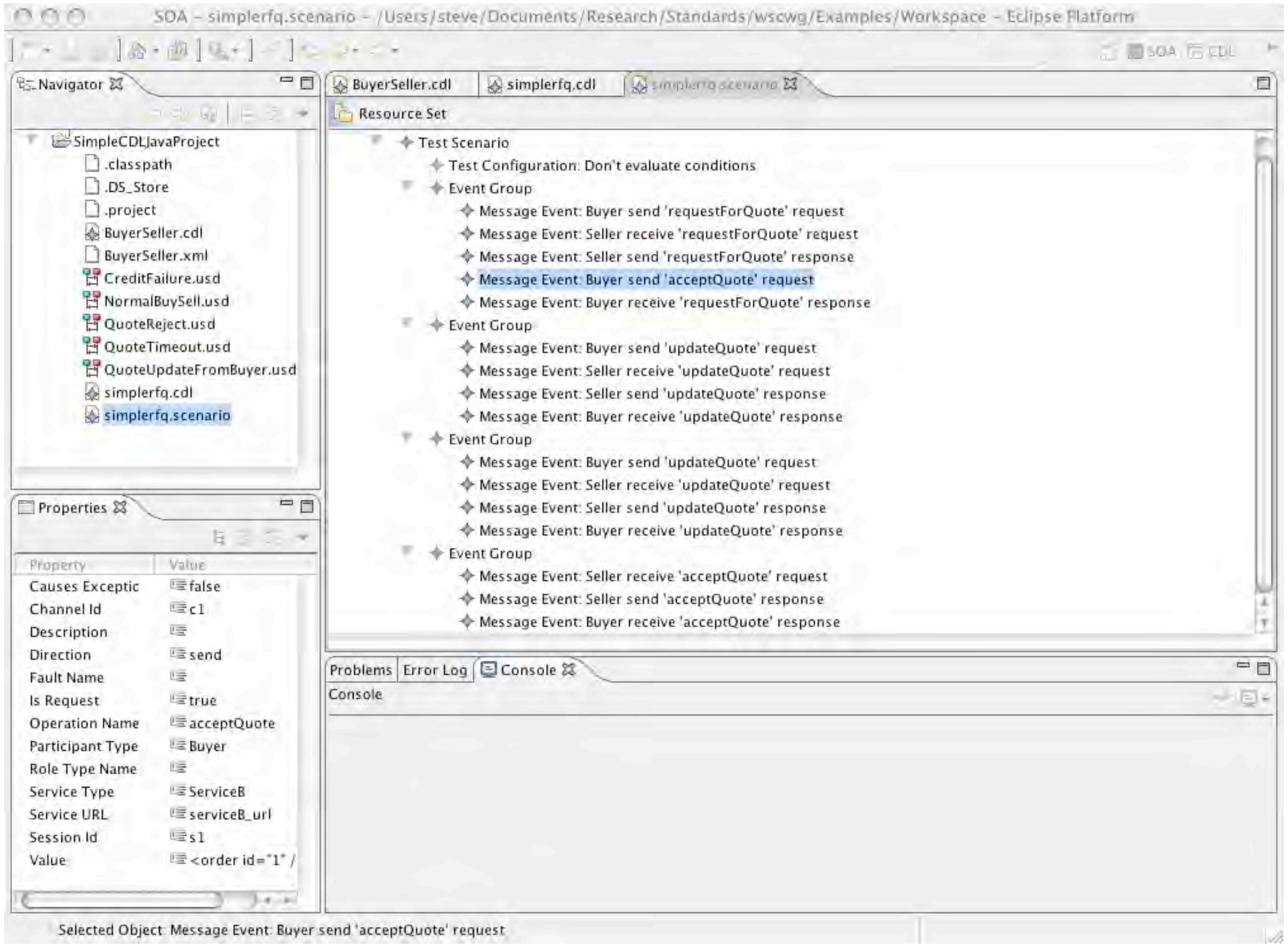
```
SimpleCDLJavaProject
    .classpath
    .DS_Store
    .project
    BuyerSeller.cdl
    BuyerSeller.xml
    CreditFailure.usd
    NormalBuySell.usd
    QuoteReject.usd
    QuoteTimeout.usd
    QuoteUpdateFromBuyer.usd
    simplerfq.cdl
    simplerfq.scenario
```

```
platform:/resource/SimpleCDLJavaProject/simplerfq.scenario
  Test Scenario
    Test Configuration: Don't evaluate conditions
    Event Group
      Message Event: Buyer send 'requestForQuote' request
      Message Event: Seller receive 'requestForQuote' request
      Message Event: Seller send 'requestForQuote' response
      Message Event: Buyer receive 'requestForQuote' response
    Event Group
      Message Event: Buyer send 'updateQuote' request
      Message Event: Seller receive 'updateQuote' request
      Message Event: Seller send 'updateQuote' response
      Message Event: Buyer receive 'updateQuote' response
    Event Group
      Message Event: Buyer send 'updateQuote' request
      Message Event: Seller receive 'updateQuote' request
      Message Event: Seller send 'updateQuote' response
      Message Event: Buyer receive 'updateQuote' response
    Event Group
      Message Event: Buyer send 'acceptQuote' request
      Message Event: Seller receive 'acceptQuote' request
      Message Event: Seller send 'acceptQuote' response
      Message Event: Buyer receive 'acceptQuote' response
```

Property	Value
Causes Exception	false
Channel Id	c1
Description	
Direction	send
Fault Name	
Is Request	true
Operation Name	acceptQuote
Participant Type Name	Buyer
Role Type Name	
Service Type	ServiceB
Service URL	serviceB_url
Session Id	s1
Value	<order id="1" />

Selected Object: Message Event: Buyer send 'acceptQuote' request

WS-CDL Bad Test Script



SOA – simplerfq.scenario – /Users/steve/Documents/Research/Standards/wscwg/Examples/Workspace – Eclipse Platform

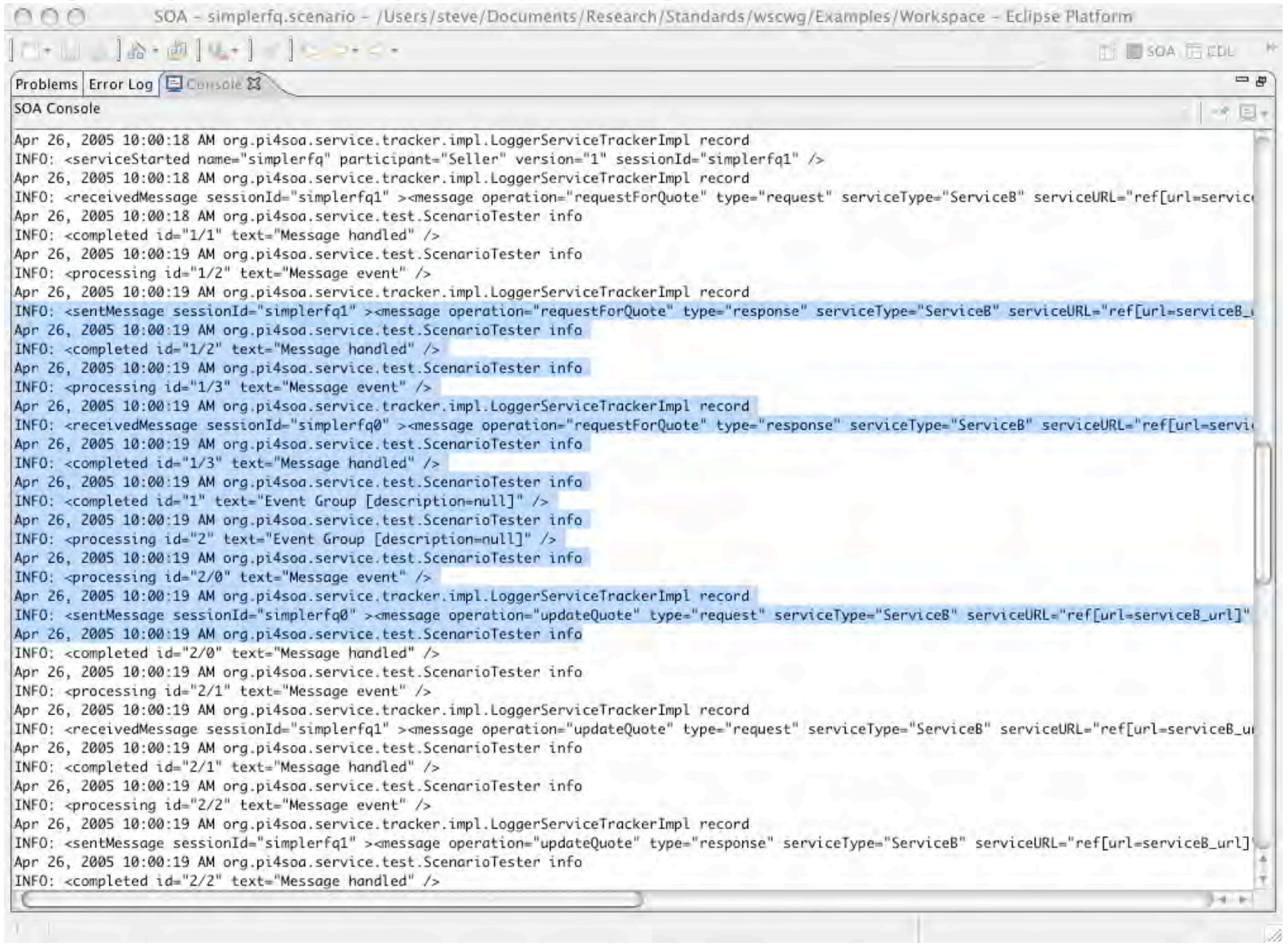
```
SimpleCDLJavaProject
    .classpath
    .DS_Store
    .project
    BuyerSeller.cdl
    BuyerSeller.xml
    CreditFailure.usd
    NormalBuySell.usd
    QuoteReject.usd
    QuoteTimeout.usd
    QuoteUpdateFromBuyer.usd
    simplerfq.cdl
    simplerfq.scenario
```

```
Test Scenario
    Test Configuration: Don't evaluate conditions
    Event Group
        Message Event: Buyer send 'requestForQuote' request
        Message Event: Seller receive 'requestForQuote' request
        Message Event: Seller send 'requestForQuote' response
        Message Event: Buyer send 'acceptQuote' request
        Message Event: Buyer receive 'requestForQuote' response
    Event Group
        Message Event: Buyer send 'updateQuote' request
        Message Event: Seller receive 'updateQuote' request
        Message Event: Seller send 'updateQuote' response
        Message Event: Buyer receive 'updateQuote' response
    Event Group
        Message Event: Buyer send 'updateQuote' request
        Message Event: Seller receive 'updateQuote' request
        Message Event: Seller send 'updateQuote' response
        Message Event: Buyer receive 'updateQuote' response
    Event Group
        Message Event: Seller receive 'acceptQuote' request
        Message Event: Seller send 'acceptQuote' response
        Message Event: Buyer receive 'acceptQuote' response
```

Property	Value
Causes Exceptic	false
Channel Id	c1
Description	
Direction	send
Fault Name	
Is Request	true
Operation Name	acceptQuote
Participant Type	Buyer
Role Type Name	
Service Type	ServiceB
Service URL	serviceB_url
Session Id	s1
Value	<order id="1" /

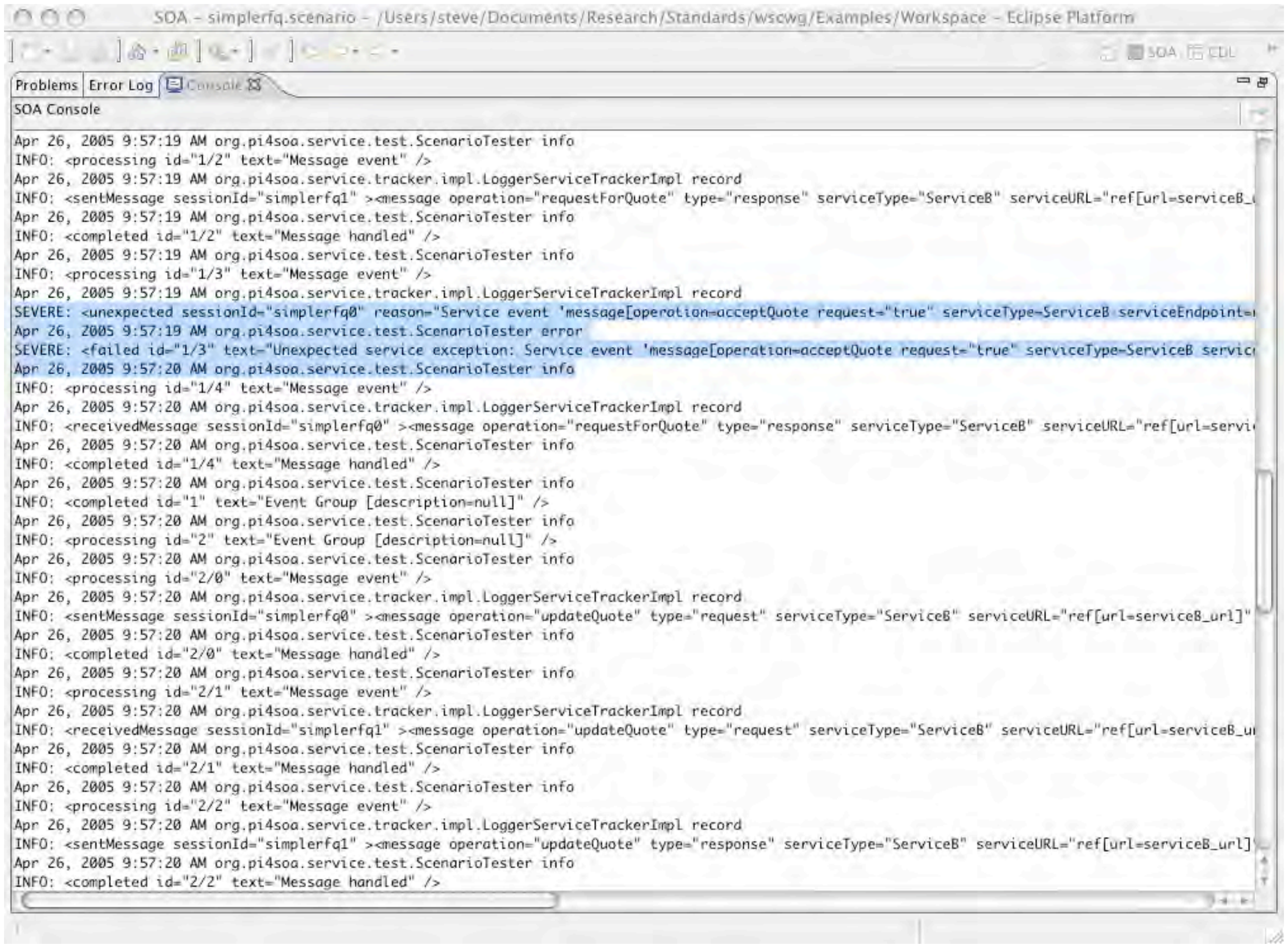
Selected Object: Message Event: Buyer send 'acceptQuote' request

WS-CDL Good Simulation


SOA – simplerfq.scenario – /Users/steve/Documents/Research/Standards/wscwg/Examples/Workspace – Eclipse Platform
SOA
CDL
Problems
Error Log
Console
SOA Console

```
Apr 26, 2005 10:00:18 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <serviceStarted name="simplerfq" participant="Seller" version="1" sessionId="simplerfq1" />
Apr 26, 2005 10:00:18 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <receivedMessage sessionId="simplerfq1" ><message operation="requestForQuote" type="request" serviceType="ServiceB" serviceURL="ref[url=servic
Apr 26, 2005 10:00:18 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="1/1" text="Message handled" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="1/2" text="Message event" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <sentMessage sessionId="simplerfq1" ><message operation="requestForQuote" type="response" serviceType="ServiceB" serviceURL="ref[url=serviceB_
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="1/2" text="Message handled" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="1/3" text="Message event" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <receivedMessage sessionId="simplerfq0" ><message operation="requestForQuote" type="response" serviceType="ServiceB" serviceURL="ref[url=servi
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="1/3" text="Message handled" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="1" text="Event Group [description=null]" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="2" text="Event Group [description=null]" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="2/0" text="Message event" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <sentMessage sessionId="simplerfq0" ><message operation="updateQuote" type="request" serviceType="ServiceB" serviceURL="ref[url=serviceB_url]"
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="2/0" text="Message handled" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="2/1" text="Message event" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <receivedMessage sessionId="simplerfq1" ><message operation="updateQuote" type="request" serviceType="ServiceB" serviceURL="ref[url=serviceB_u
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="2/1" text="Message handled" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="2/2" text="Message event" />
Apr 26, 2005 10:00:19 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <sentMessage sessionId="simplerfq1" ><message operation="updateQuote" type="response" serviceType="ServiceB" serviceURL="ref[url=serviceB_url]
Apr 26, 2005 10:00:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="2/2" text="Message handled" />
```

WS-CDL Bad Simulation


SOA – simplerfq.scenario – /Users/steve/Documents/Research/Standards/wscwg/Examples/Workspace – Eclipse Platform
SOA
CDL
Problems
Error Log
Console
SOA Console

```
Apr 26, 2005 9:57:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="1/2" text="Message event" />
Apr 26, 2005 9:57:19 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <sentMessage sessionId="simplerfq1" ><message operation="requestForQuote" type="response" serviceType="ServiceB" serviceURL="ref[url=serviceB_
Apr 26, 2005 9:57:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="1/2" text="Message handled" />
Apr 26, 2005 9:57:19 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="1/3" text="Message event" />
Apr 26, 2005 9:57:19 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
SEVERE: <unexpected sessionId="simplerfq0" reason="Service event 'message[operation=acceptQuote request="true" serviceType=ServiceB serviceEndpoint=
Apr 26, 2005 9:57:19 AM org.pi4soa.service.test.ScenarioTester error
SEVERE: <failed id="1/3" text="Unexpected service exception: Service event 'message[operation=acceptQuote request="true" serviceType=ServiceB servic
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="1/4" text="Message event" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <receivedMessage sessionId="simplerfq0" ><message operation="requestForQuote" type="response" serviceType="ServiceB" serviceURL="ref[url=servi
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="1/4" text="Message handled" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="1" text="Event Group [description=null]" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="2" text="Event Group [description=null]" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="2/0" text="Message event" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <sentMessage sessionId="simplerfq0" ><message operation="updateQuote" type="request" serviceType="ServiceB" serviceURL="ref[url=serviceB_url]"
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="2/0" text="Message handled" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="2/1" text="Message event" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <receivedMessage sessionId="simplerfq1" ><message operation="updateQuote" type="request" serviceType="ServiceB" serviceURL="ref[url=serviceB_u
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="2/1" text="Message handled" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <processing id="2/2" text="Message event" />
Apr 26, 2005 9:57:20 AM org.pi4soa.service.tracker.impl.LoggerServiceTrackerImpl record
INFO: <sentMessage sessionId="simplerfq1" ><message operation="updateQuote" type="response" serviceType="ServiceB" serviceURL="ref[url=serviceB_url]
Apr 26, 2005 9:57:20 AM org.pi4soa.service.test.ScenarioTester info
INFO: <completed id="2/2" text="Message handled" />
```

Summary

- **CDL is a language for describing observable behaviour of multiple participants within the context of a global model**
  - Formally based on pi-calculus
  - Useful outside of Web Services
  - Robustness, conformance, testing, verification
- **Complimentary to many areas**
  - BPEL, Agents, FIX, fpML, TWIST, etc.

Resources

- **W3C WS-Choreography WG**
  - http://www.w3.org/2002/ws/chor/
- **Pi Calculus for SOA**
  - http://www.pi4soa.org/
- **Pi4 Technologies**
  - http://www.pi4tech.com/
- **My Info**
  - Charlton Barreto
    cbarreto@adobe.com
    http://www.adobe.com/

Demo

Adobe®