Web Services Choreography

Ing. Enrico Oliva

PhD Student

{eoliva@deis.unibo.it}

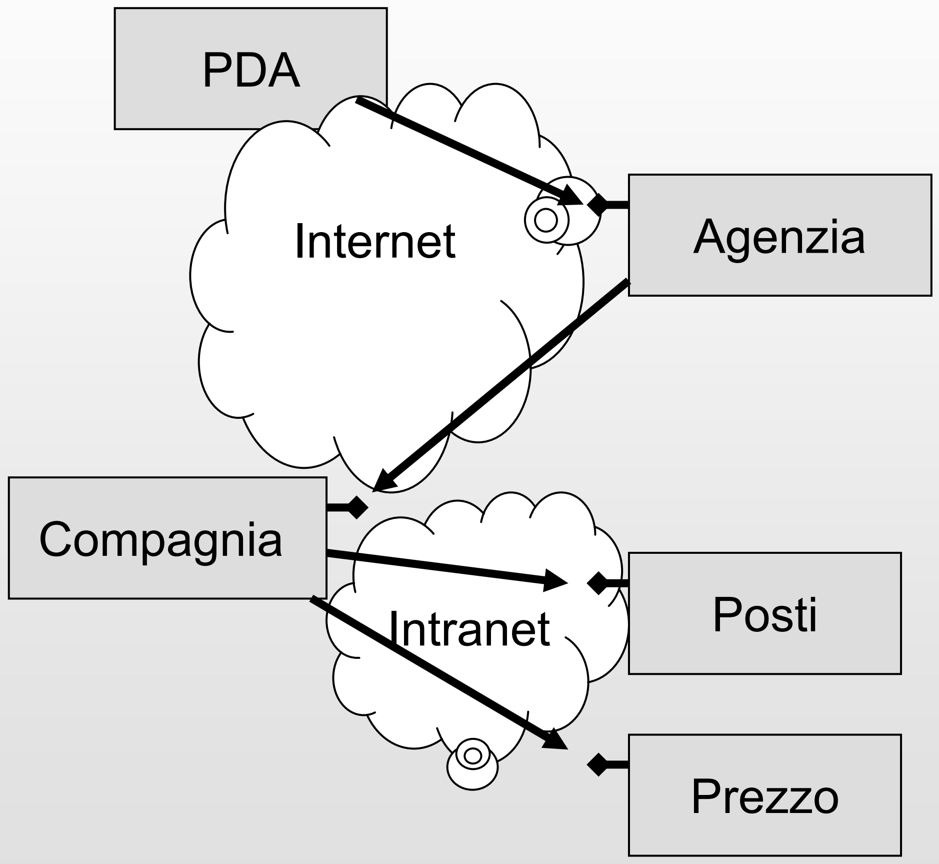
Outline

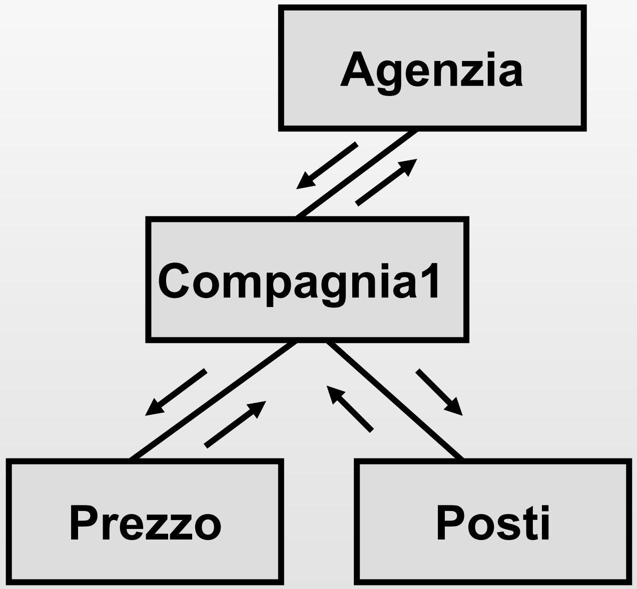
- Choreography & Orchestration
- Orchestration with WS-BPEL
- Choreography with WS-CDL
  - Why WS-CDL?
  - What is WS-CDL?
  - Where is WS-CDL?
- An example of choreography between buyer, seller, credit agency and shipper
  - Bubble and stick, Sequence Diagrams and WS-CDL
- WS-CDL Approach
  - Why it is based on Pi-Calculus?
- WS-CDL tool: Pi4SOA
- STIL project
  - Design with WS-CDL the service decomposition realized by SATA
- Some pictures and ideas taken from presentation of Steve Ross Talbot - Pi4 Technologies

Choreography & Orchestration

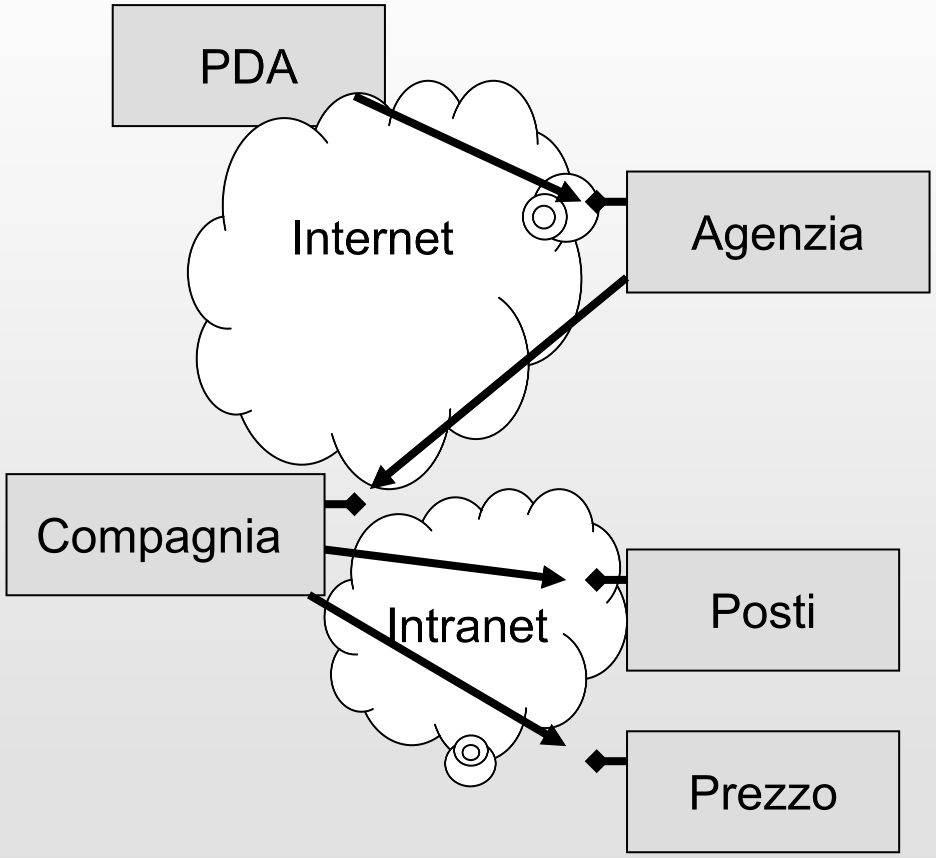
- Choreography is a peer to peer interaction in a global model, it does not depend on a centralized controller
  - It is about describing and guiding a global model
  - You can derive the single viewpoint model from the global model by a projection
- Orchestration is a hierarchical request/provider model, it implies a centralized control mechanism
  - It defines what and when the services should be called but it does not define a collaboration among multi parties
  - It is about describing and executing a single viewpoint model

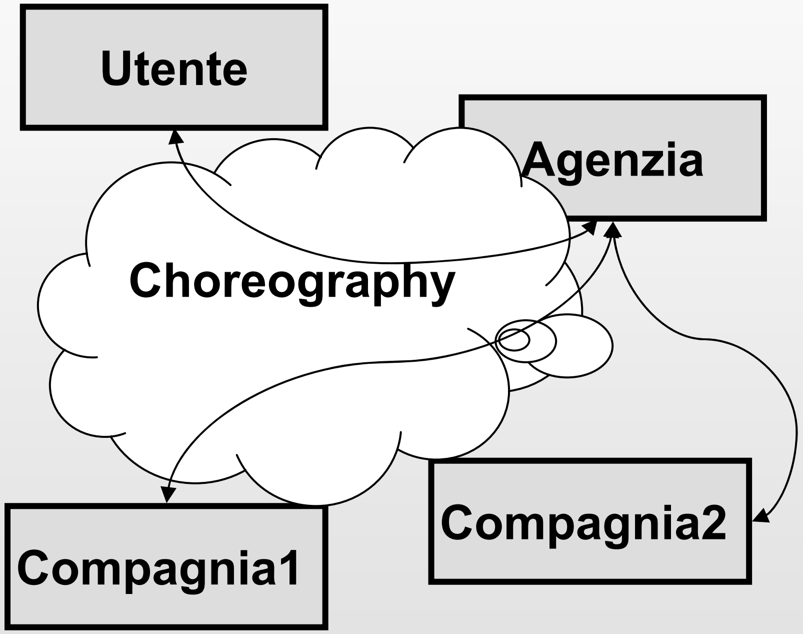
Orchestration





Choreography

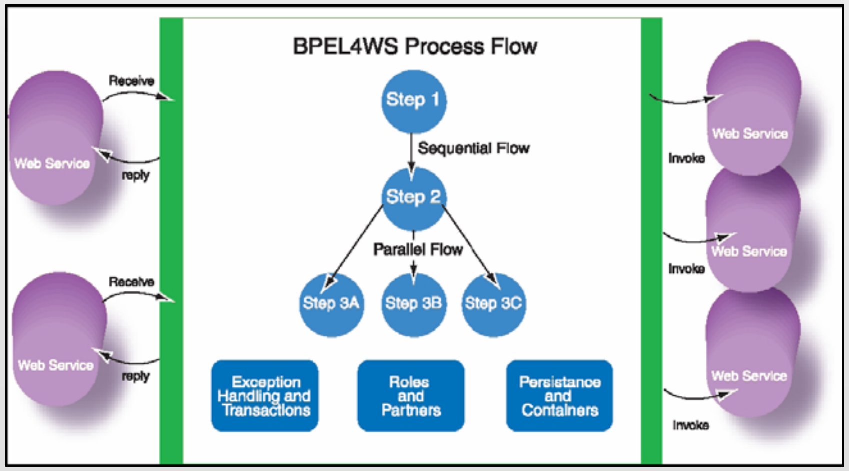




Orchestration with WS-BPEL

- Web Services – Business Process Execution Language (BPEL or WS-BPEL) is a process-oriented composition language for Web services
  - It relies on WSDL
  - Structures: sequence, fork, join, parallel threads, computation
  - A BPEL process is a Web service with WSDL interface
  - Implies a centralized control mechanism

- A BPEL process executes the necessary WSDL calls by effecting message exchange between services
- A BPEL process can invoke another BPEL process and it can call itself recursively



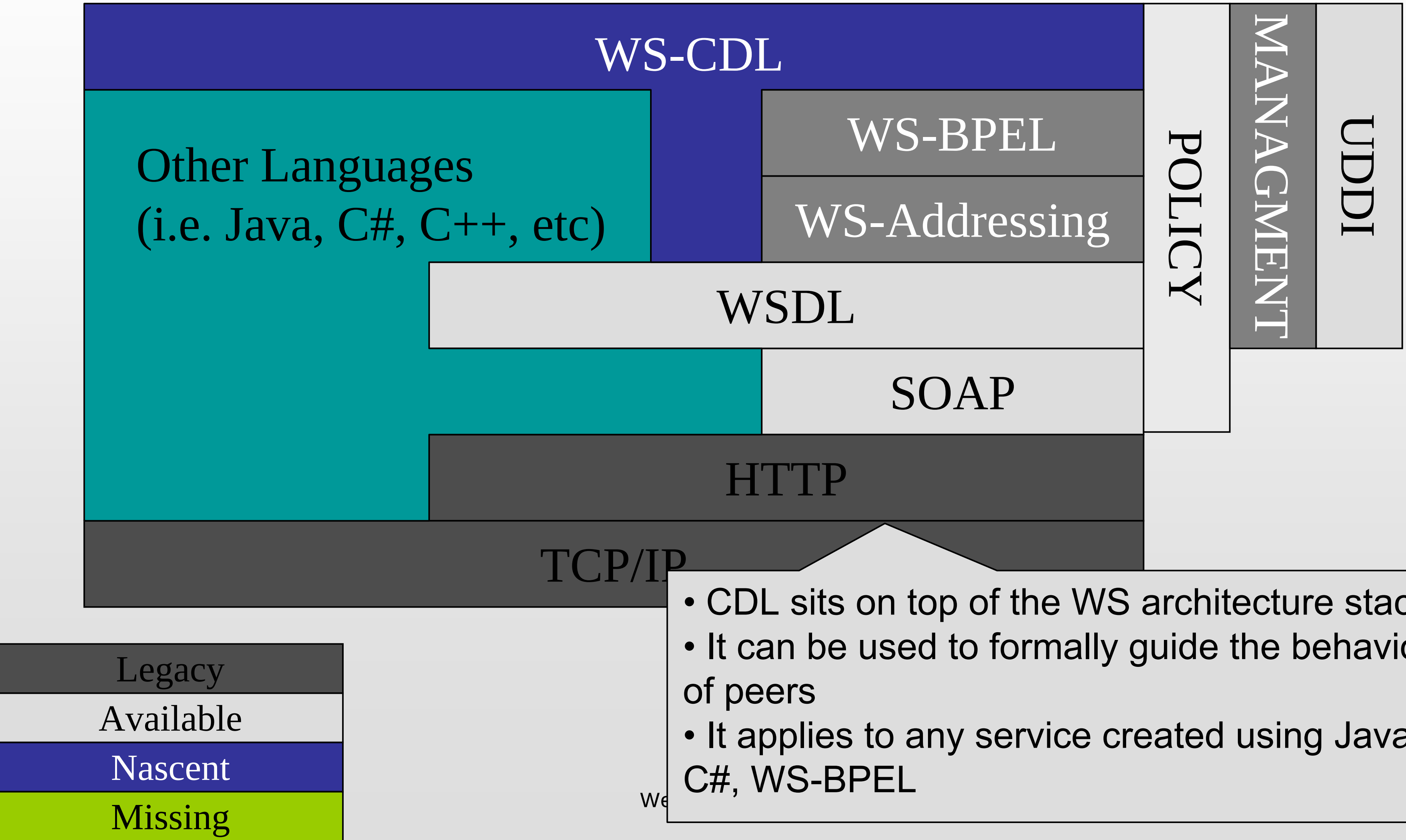
Why a Choreography Language?

- *Each service can be described using WSDL or some other interface languages (ex. Java)*
  - *But this specification does not provide the sequence and the conditions of the calls*
- A language for the business activity that involves different organizations is necessary, describing the **collaboration** between the processes in a scalable and unambiguous way

What is WS-CDL?

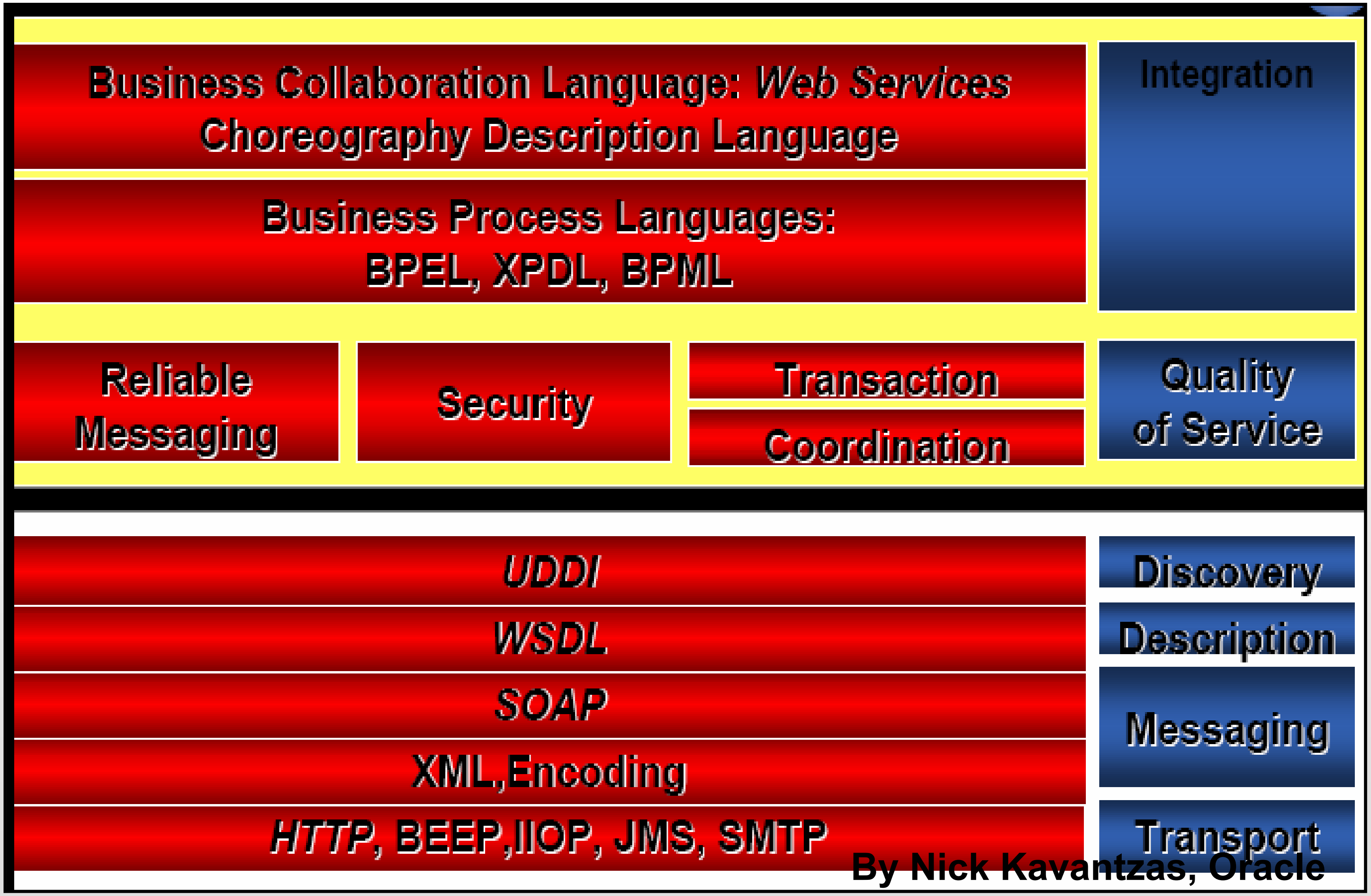
- WS-CDL is the Web Services Choreography Description Language (CDL for short)
- It is a language that can be used to describe collaboration protocols of cooperating [Web] Service participants in which
  - Services act as peers
  - Interactions may be long-lived and statefull
- A CDL-based description is a multi-participant contract that describes, from a neutral or global viewpoint, the *common* observable behavior (ex. WSDL, Java interface) of the collaborating Service participants
  - The observable behavior is the behavior of a service which can be observed without looking inside to see how the service is doing things

Where is WS-CDL?



- CDL sits on top of the WS architecture stack
- It can be used to formally guide the behavior of peers
- It applies to any service created using Java, C#, WS-BPEL

Emerging Web Services platform



Business Collaboration Language: *Web Services Choreography Description Language*
Business Process Languages: BPEL, XPDL, BPML
Integration
Reliable Messaging
Security
Transaction
Coordination
Quality of Service
UDDI
WSDL
SOAP
XML,Encoding
HTTP, BEEP,IIOP, JMS, SMTP
Discovery
Description
Messaging
Transport

By Nick Kavantzas, Oracle

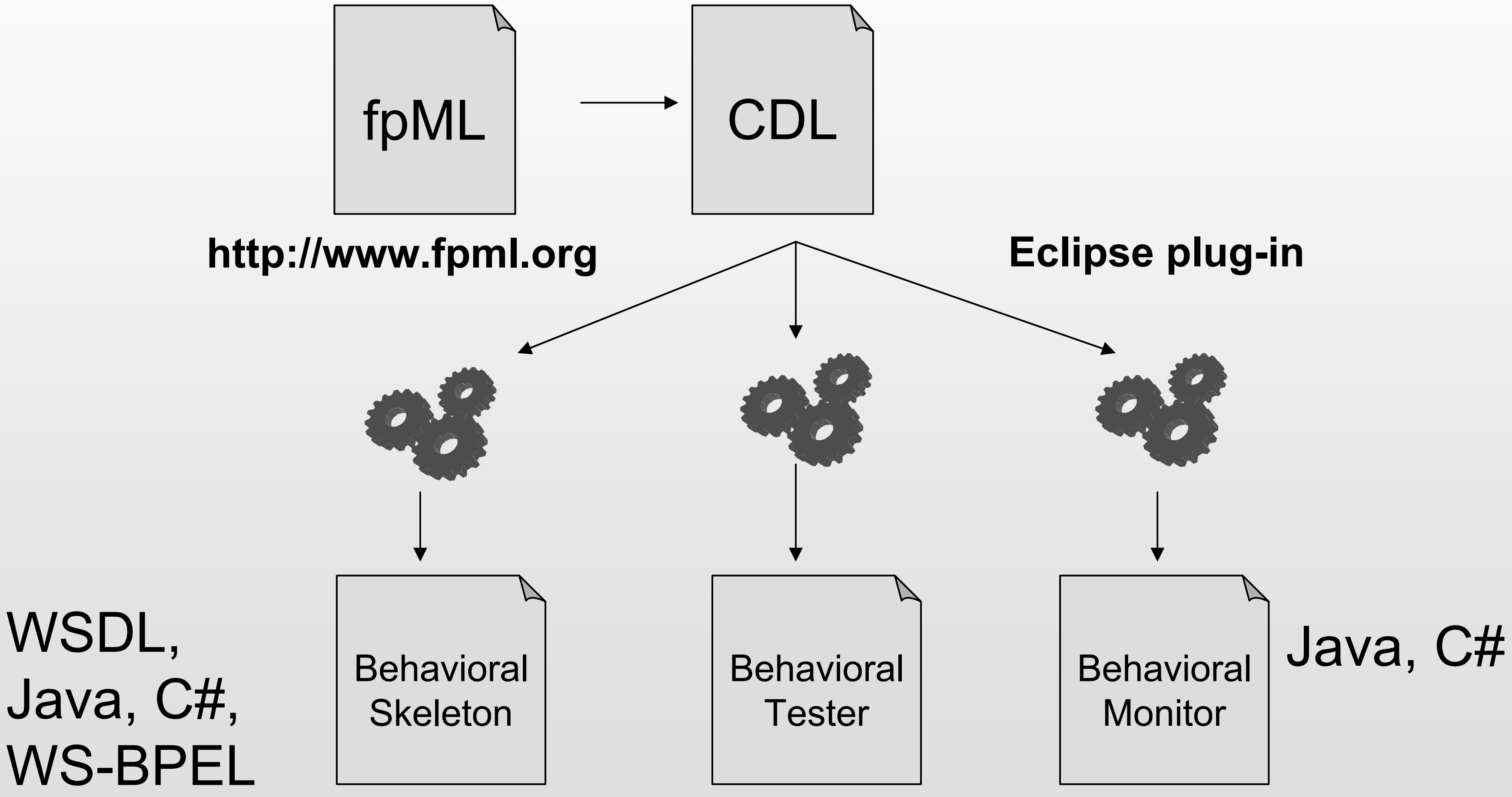
WS-CDL vs WS-BPEL

- WS-BPEL
  - Executable language (also for abstract processes)
  - Recursive Web Service Composition
  - Centralised control by orchestration service
  - Based on BPEL4WS1.1
- WS-CDL
  - Description language
  - Multi-party contracts (blueprints) for services as peers
  - No centralized control, control is shared between domains
  - Does not need Web Services but is targeted to deliver over them
  - WS-CDL doesn’t see WS-BPEL is unique or different to any other end-point language target

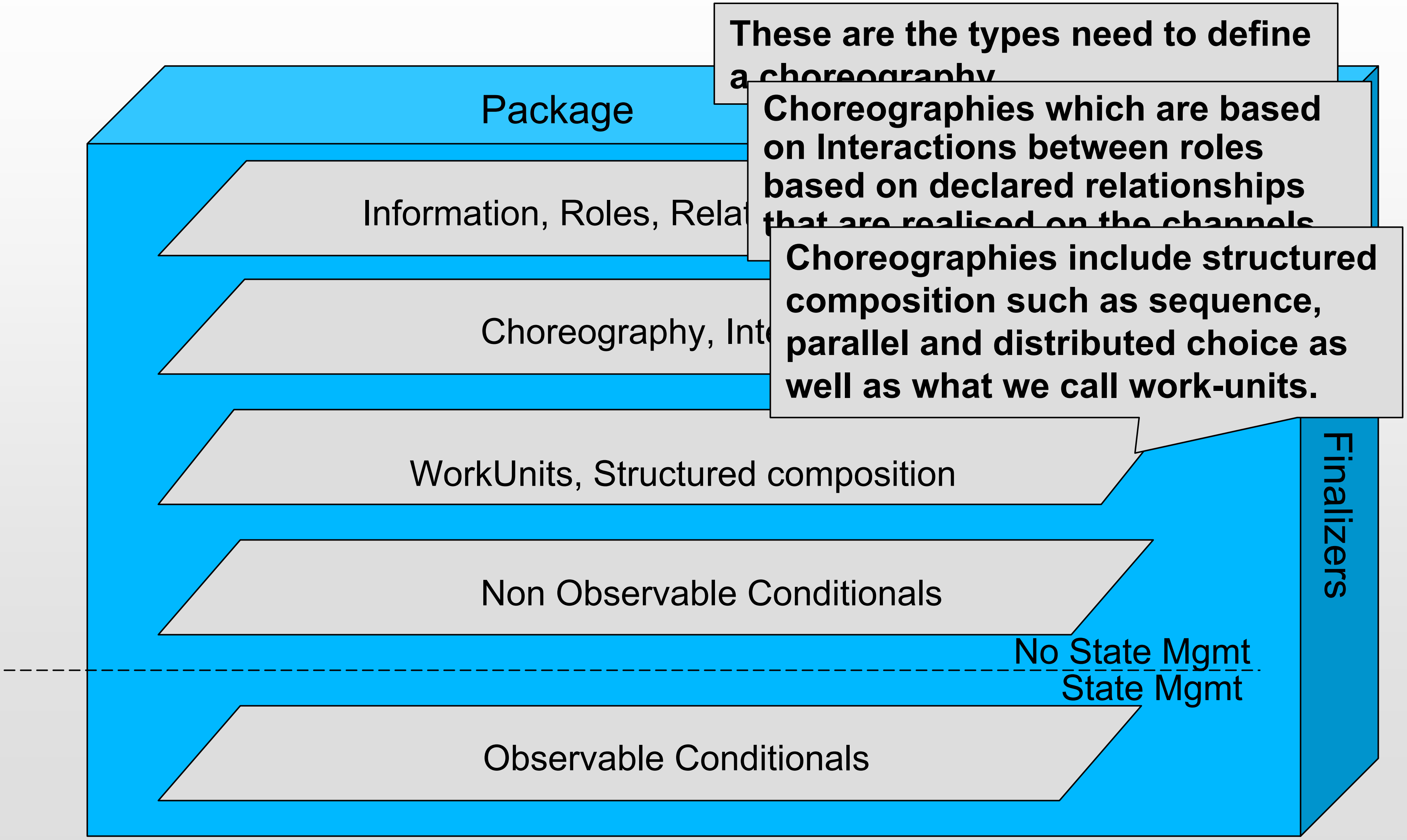
Why would I use CDL?

- To **ensure effective interoperability** of Services is guaranteed because Services will have to conform to a common behavioral multi-party contract specified in the CDL
- To create **more robust Services** because they can be validated statically and at runtime against a choreography description
- To **reduce the cost of implementing** Services by ensuring conformance to expected behaviour
- To **ensures that collaborative development can delivery**

How would I use it?



WS-CDL Structure


Package
Information, Roles, Relat
Choreography, Inte
WorkUnits, Structured composition
Non Observable Conditionals
No State Mgmt
State Mgmt
Observable Conditionals
Finalizers
These are the types need to define a choreography
Choreographies which are based on Interactions between roles based on declared relationships that are realised on the channels
Choreographies include structured composition such as sequence, parallel and distributed choice as well as what we call work-units.

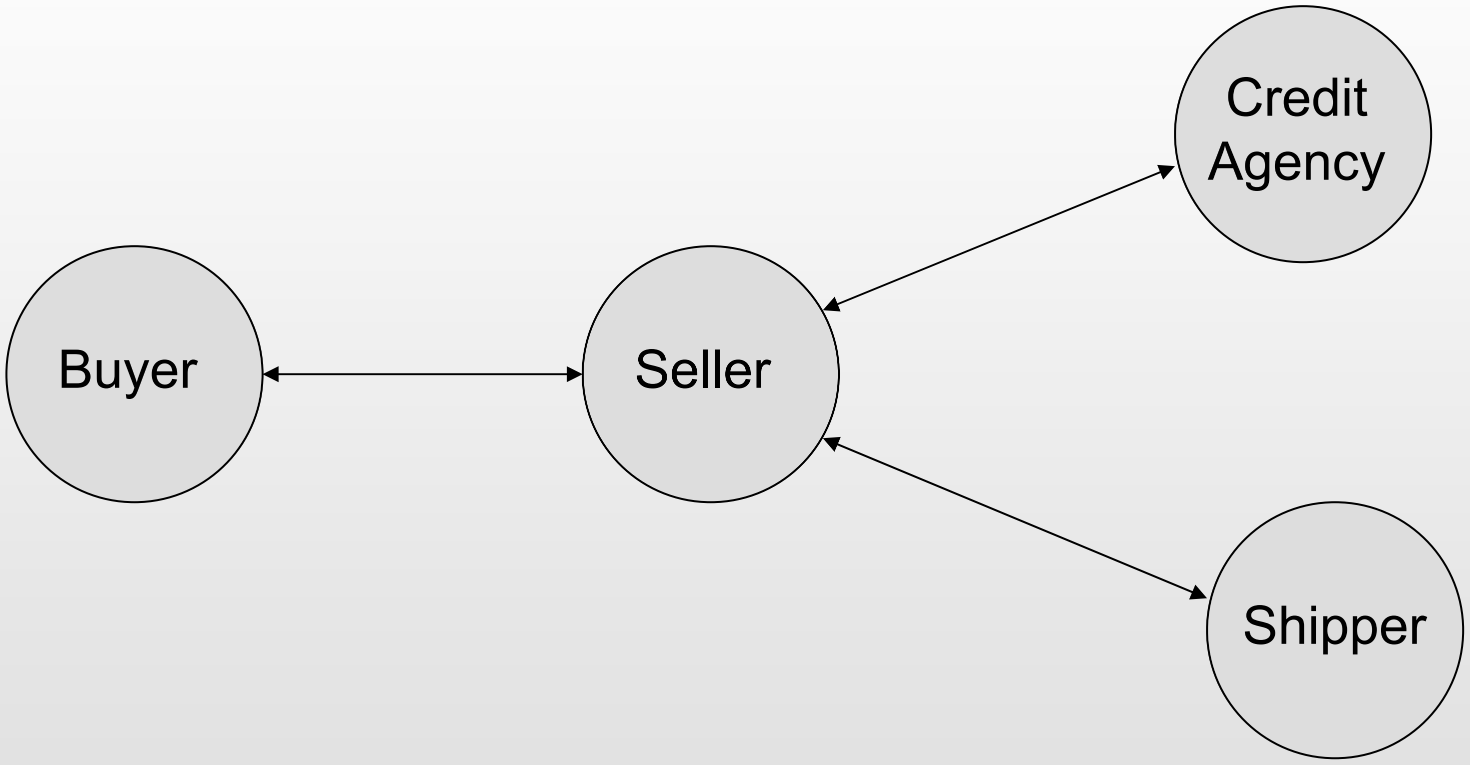
An Example

- Actors
  - Buyer, Seller, Credit Agency, Shipper
- Actions
  - Buyer barters with the Seller to get a price
  - Buyer accepts a price and places an order
  - Seller checks Buyers credit worthiness
  - Seller requests delivery from Shipper
  - Shipper sends delivery details to Seller and to Buyer

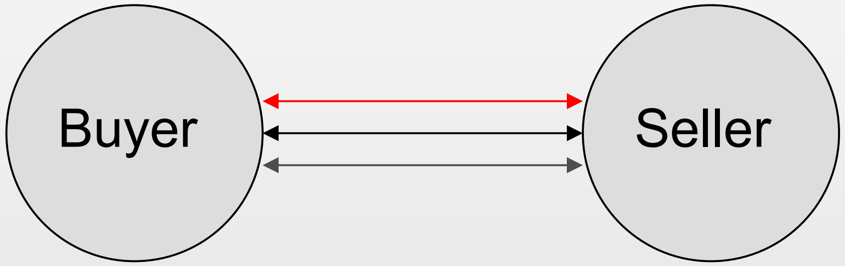
- **Bubble and stick**
- **Sequence diagrams**
- **Activity diagrams**
  - **Interaction Overview diagrams (UML 2.0)**

→ CDL

Bubble and Stick



Bubble and Stick



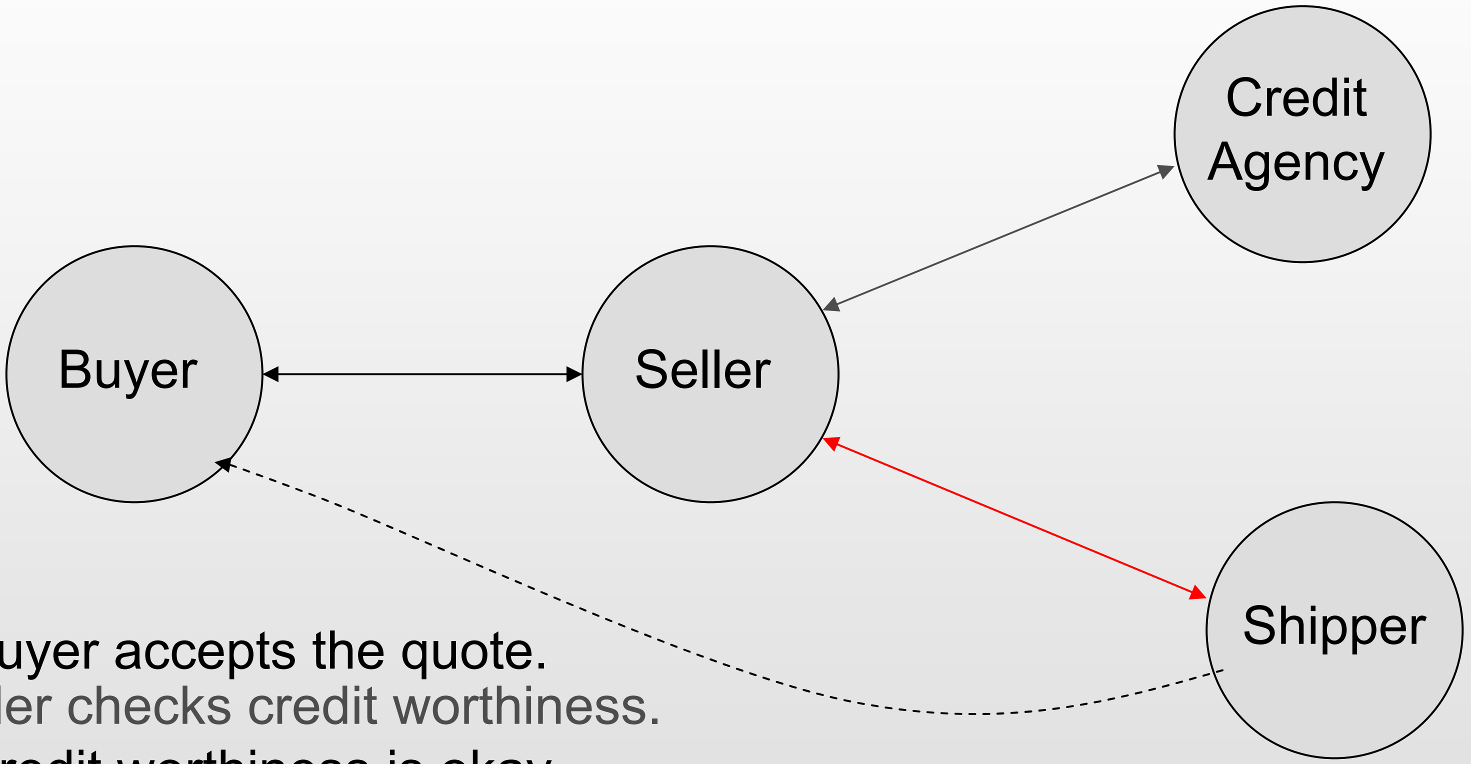
- Buyer request a quote from the seller.
- Seller responds with a quote.

- Buyer MAY accept the quote.

- Buyer MAY update quote and request the update from the seller.
- Seller MAY respond with the update quote.

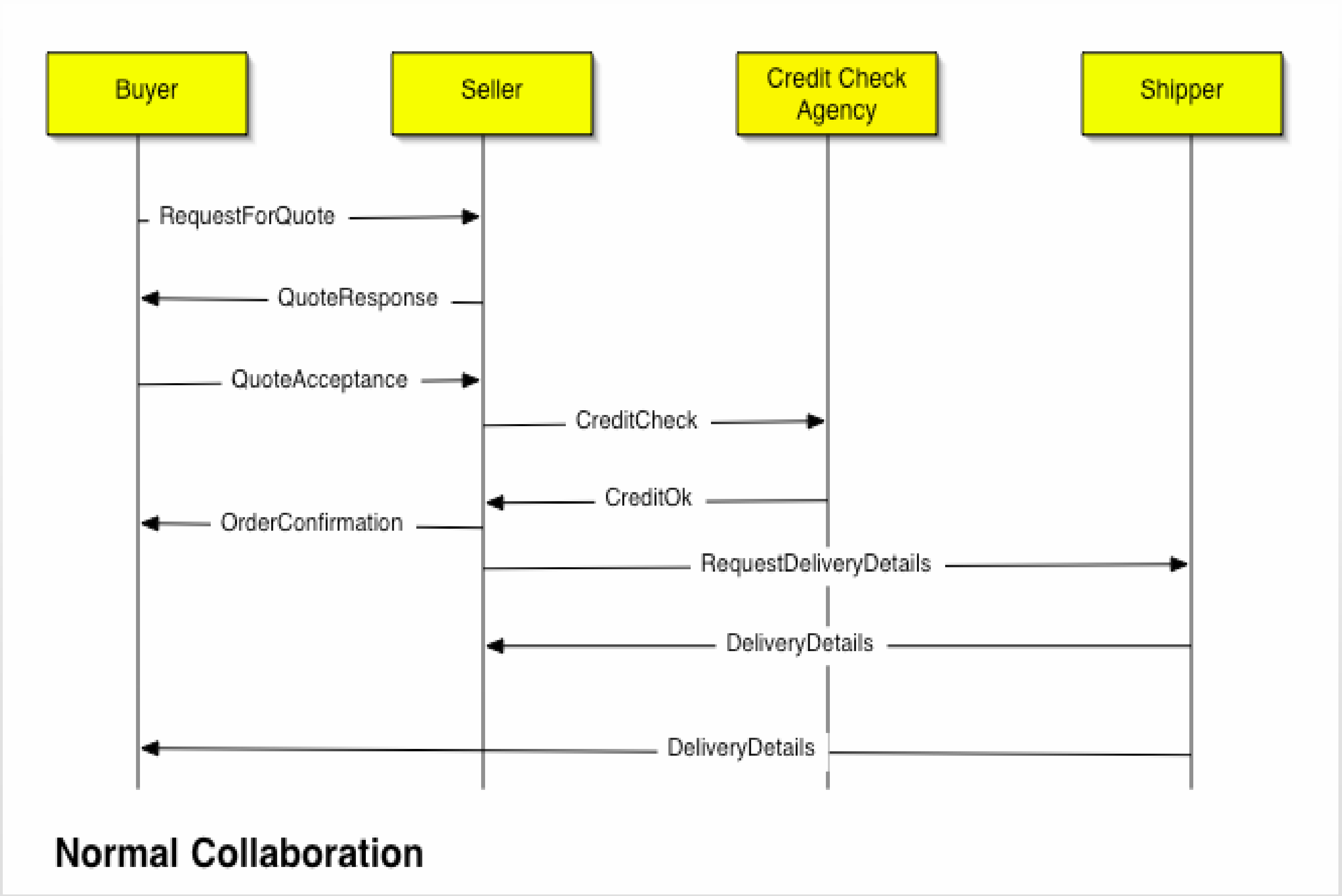
- Quotes may timeout.

Bubble and Stick

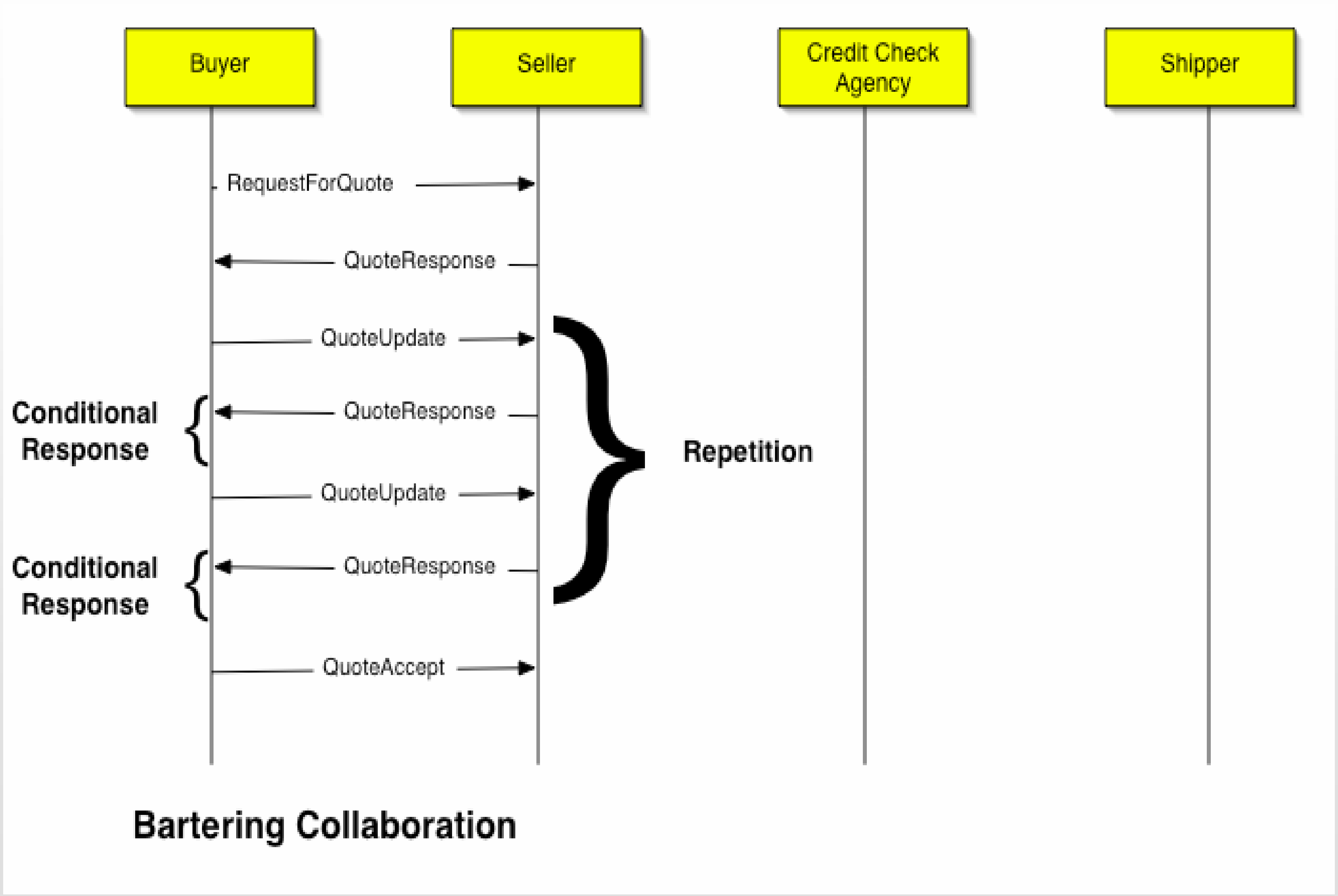


- If Buyer accepts the quote.
- Seller checks credit worthiness.
- If Credit worthiness is okay
- Seller requests delivery from Shipper.
- Shipper sends delivery details back to Seller and to Buyer.

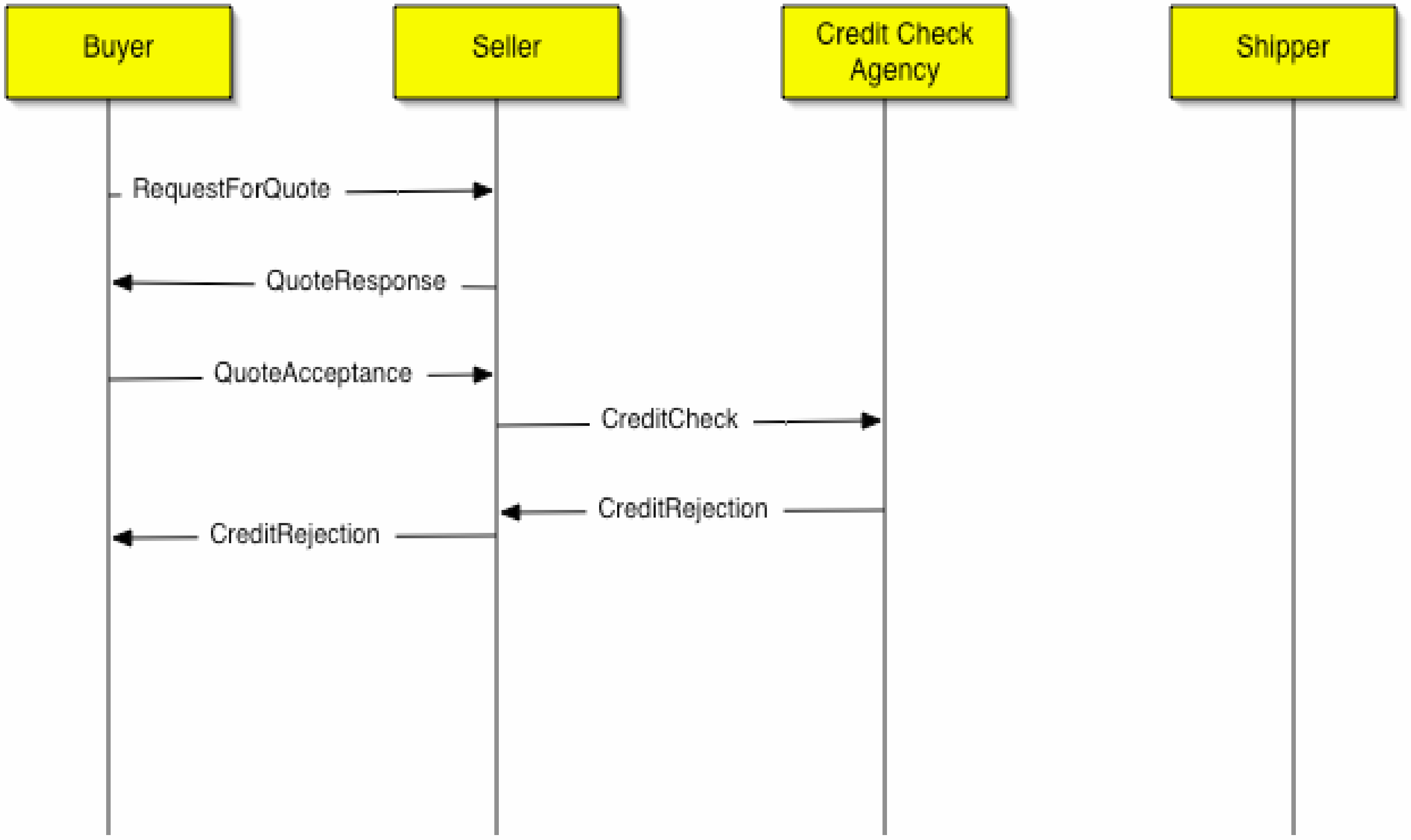
Sequence Diagrams



Sequence Diagrams

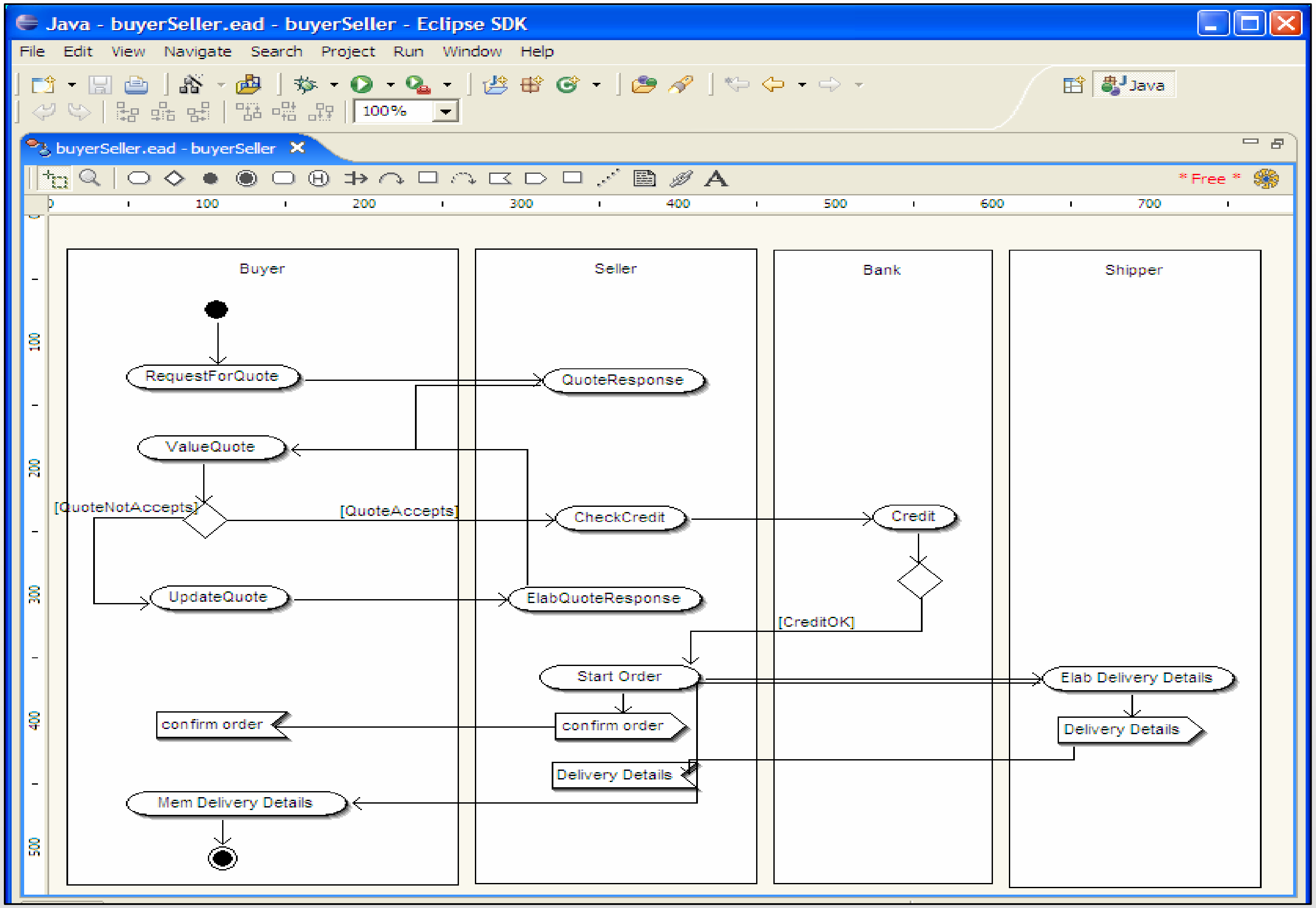


Sequence Diagrams

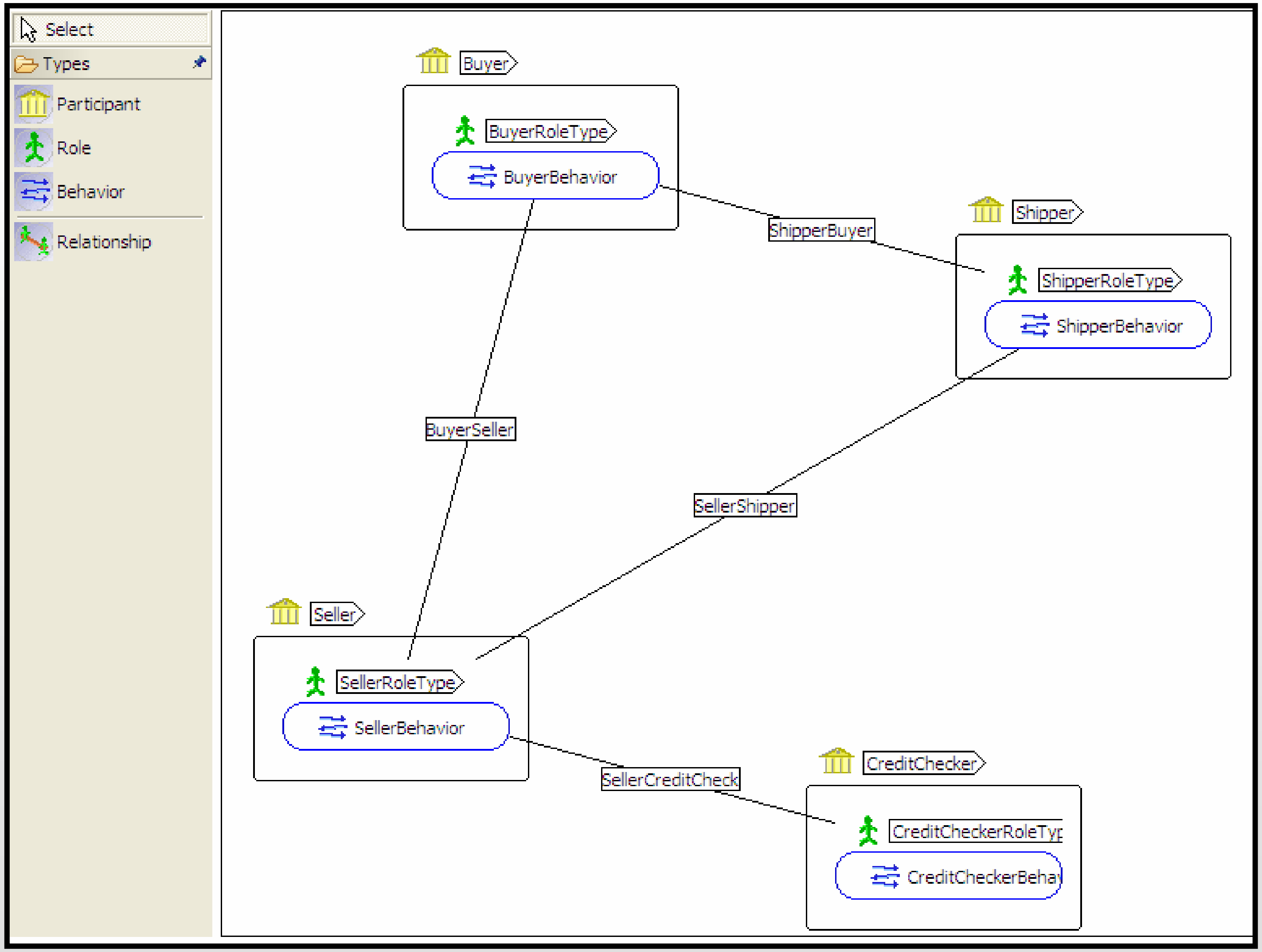


Credit Rejection Collaboration

Activity Diagrams



WS-CDL



Select
Types
Participant
Role
Behavior
Relationship
Buyer
BuyerRoleType
BuyerBehavior
ShipperBuyer
Shipper
ShipperRoleType
ShipperBehavior
BuyerSeller
SellerShipper
Seller
SellerRoleType
SellerBehavior
SellerCreditCheck
CreditChecker
CreditCheckerRoleTyp
CreditCheckerBeha

WS-CDL 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 Zilio; Yoshida, et al)
  - Race-condition detection (refs)
- Which are extensions to CCS/CSP and π-calulus (Milner)

WS-CDL Approach

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

WS-CDL and the Pi-Calculus

- Pi-calculus is a language used to define concurrent processes that interact with one another dynamically
  - The most distinct feature is mobility
    - The topology of communicating processes changes dynamically in response to channel passing
- Choreography has to build global collaborative contracts requiring a conceptual framework that can express dynamic communicating processes precisely and concisely
  - WS-CDL based its constructs on the Pi Calculus

WS-CDL and the Pi-Calculus

Operation	Notation	Meaning
Prefix	**$\pi.P$**	**Sequence**
Action	**$a(y)$, $\overline{a}(y)$**	
Summation	**$a(y).P + b(x).Q$** $\sum \pi_{i(}P_i$	**Choice**
Recursion	**$P=\{\ldots\ldots\}.P$**	**Repetition**
Replication	**$!P$**	**Repetition**
Composition	**$P \mid Q$**	**Concurrency**
Restriction	**$(\nu x)P$**	**Encapsulation**

Collapse send and receive into an interact on channels

WS-CDL Concepts & Pi-Calculus

- Central concepts in WS-CDL are interaction, channel and guarded workUnit
  - A **channel** represents a pair of “ports” in pi-calculus
    - They represent a declared name binding of ports between process
  - A **interaction** is a message exchange that occurs in a channel
    - The message may be represented in pi-calculus as a polyadic message
    - The channel and their interaction enable a bi-directional communication, modelling a request and response pair
    - The type of messages exchange can be represented as “sorts” in pi-calculus
  - A **guarded workUnit** waits until a condition is met
    - The workUnit may be represented in pi-calculus as a process or collection of process where each component in the condition is a port with a condition attached

WS-CDL Formalisms

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

Syntax:

Inf set N of names x,y,u and literals, x means x_1 .. x_n (n>=0), loc means locations

Process P, Q, E, F ::=

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

Guard g,h ::=

loc:u | loc: u # v | g + g | g g | h + h | h h

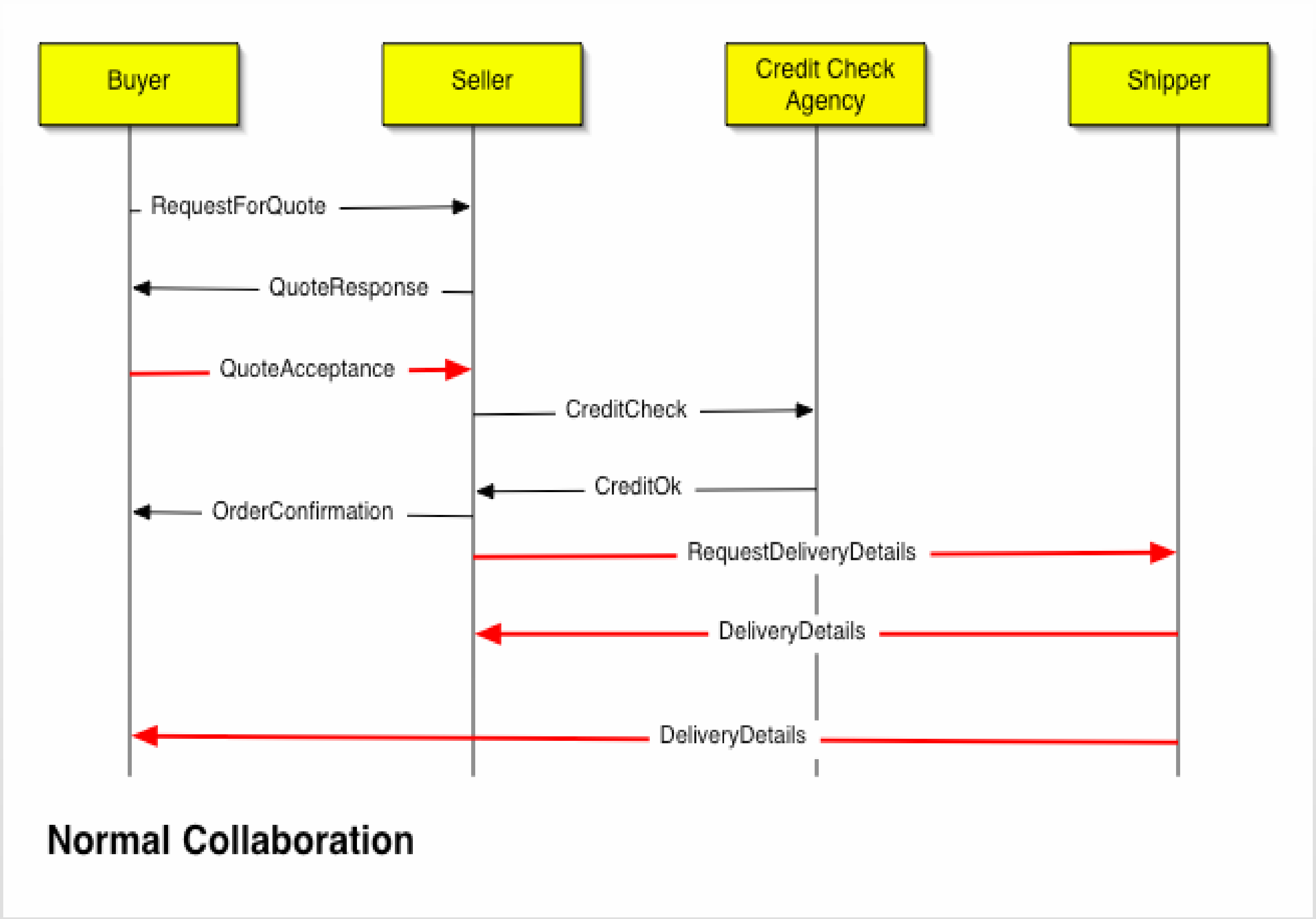
Outline

- Choreography & Orchestration
- Orchestration with WS-BPEL
- Choreography with WS-CDL
- An example of choreography between buyer, seller, credit agency and shipper
  - Bubble and stick, Sequence Diagrams and WS-CDL
- WS-CDL Approach
  - Why it is based on Pi-Calculus?

➢ **WS-CDL in Detail**
  - ➢ **Syntax**
  - ➢ **Implementation**

➢ **WS-CDL tool: Pi4SOA**

Sequence Diagrams



Typing

- Information type
  - Aliases WSDL type, 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 Locater type
  - Specify rules for selecting a piece of information within a document

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

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

<token  name="ncname"
    informationType="qname"
    query="XPath-expression"? />
```

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>
```

- **It describe the type of information used within a Choreography**
- **The information is described as a WSDL or XML Schema**

Token Types

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

Interactions

- Enable collaborating participant to communicate and align the information
- Describe the messages exchange between two roles within a relationship along a channel istance
  - Request & Accept of an operation through a common channel
    - One way interaction single message is sent
    - Request/response interaction two message are exchanged
  - Information flow
    - request/response direction
  - State recording at roles
    - Create new, modify existing variables at a Role
  - Information Alignment
    - State changes of variables that reside in one Role with the state changes of variables that reside in the other Role
    - Value of the exchanged messages
- Interactions dependecies
  - Define our roleTypes, relationshipTypes, informationTypes, tokenType and channelTypes

Interaction Syntax

```
<interaction name="NCName" channelVariable="QName" operation="NCName"
   align="true"|"false"? initiate="true"|"false"? >

   <participate relationshipType="QName" fromRoleTypeRef="QName"
   toRoleTypeRef="QName“ /> <exchange name="NCName" faultName="QName"?
   informationType="QName"?|channelType="QName"? action="request"|"respond" >

            <send variable="XPath-expression"? recordReference="list of NCName"?
            causeException="QName"? />
             <receive variable="XPath-expression"? recordReference="list of NCName"?
                causeException="QName"? />
      </exchange>*
      <timeout time-to-complete="XPath-expression" fromRoleTypeRecordRef="list of
         NCName"? toRoleTypeRecordRef="list of NCName"? />?
      <record name="NCName" when="before"|"after"|"timeout" causeException="QName"?
         > <source variable="XPath-expression"? | expression="XPath-expression"? />
         <target variable="XPath-expression" />
      </record>*

</interaction>
```

Interactions

```
<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>
```

- This interaction describes the passing of another channel instance, called "**DeliveryDetailsC**". The channel is instantiated and it resides in a variable of the same name at the Buyer role.
- What the interaction does is passing the details through a channel, called "**Buyer2SellerC**" that enables the Shipper role to create an exact copy of it in a variable called "DeliveryDetailsC" that is passed onto the Shipper later on in the last interaction.

Interactions

```
<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="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>
```

• This interactions are broadly similar except they do not pass channels, they pass **InformationMessages** such as "**QuoteAcceptType**"

Interactions

```
<interaction name="Seller 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>
```

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>
```

- **Enumerate the observable behavior that a collaborating participant exhibits**
- **Behavior type specifies the operations supported**
    - **Optional WSDL interface**

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>
```

- **Specify the mutual commitments, in terms of Roles/Behavior types, two collaborating participant are required to provide**

Channel Types

- Realizes a *dynamic* point of collaboration, through which collaborating participant interact
  - Where and how communicate a message
    - Specify the *Role/Behavior* and *reference* of a collaborating participant
    - Identify an *Instance* of Role
- One o more channel(s) may be passed around from a Role to one or more other Role(s)
  - A channel types may restrict the types of channel allowed to be exchanged
  - A channel types may restrict its usage, by specifying the number of times channel can be used

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>
```

• In this example we allow to the instances of channel to pass other channels of type “**2BuyerChannelType**” (this is the type for our “**DeliveryDetailsC**” channel instance)

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>
```

Variables

- Capture instance information about objects in a collaboration
- Variable types
  - *Information Exchange Variables*: define instances of exchanged documents between Roles in an interaction
  - *State Variables*: define instances of state information at a Role
  - *Channel Variables:* define instances of channel types
- Their definitions
  - Specify the type of value a variable contains using informationType, channelType
  - Specify the Role of the collaboration participant a variable resides in

Choreography

- It defines re-usable common rules, that govern the ordering of exchanged messages and the provisioning patterns of collaborative behavior
  - Enumerating the observable behavior
  - Localize the visibility of variables
    - Using variable definitions
  - Prescribe alternative patterns of behavior
  - Enable recovery
- Choreography dependencies
  - Declare our variables
  - Declare our relationship types

```
<choreography name="ncname“
   complete="xsd:boolean XPath-
   expression"? isolation="dirty-write"|
   "dirty-read"|"serializable"?
   root="true"|"false"? >
   <relationship type="qname" />+
   variableDefinitions?
   Choreography-Notation*
   Activity-Notation
   <exception name="ncname">
          WorkUnit-Notation+
   </exception>?
   <finalizer name="ncname">
          WorkUnit-Notation
   </finalizer>?
</choreography>
```

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 without have linkage
    </description>
   </variable>
  <variable name="barteringDone"  informationType="BooleanType"  roleTypes="BuyerRoleType SellerR
   <description type="descriptio
  </variable>
 </variableDefinitions>
```

- Here are the variables and relationships definition
- We define some channel instances and a boolean variable

Choreography

```
<?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>
```

- Defining a choreography
- Interaction: Buyer requesting a price from the Seller
    - it is modeled with two exchanges (request/responce)

WorkUnit

- Information driven model, reaction rule guards a set of activities, by prescribing the constraints on information that need
- 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

```
<workunit name="ncname"
   guard="xsd:boolean XPath-
   expression"?

   repeat="xsd:boolean XPath-
   expression"? block="true|false" >

   Activity-Notation

 </workunit>
```

WorkUnit

- WorkUnit explanation with imperative language principles
  - Workunit (G) (R) (B is True) Body
    - 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

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 }

Batering 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>
```

- **Bartering process**
  - **Interaction between Buyer and Seller**
  - **Interaction to pass call back details**

Batering Process

```
      <assign roleType="BuyerRoleType">
         <copy name="copy">
            <source expression="true" />
            <target variable="cdl:getVariable('barteringDone','','')" />
         </copy>
      </assign>
     </sequence>
    <sequence>
       <interaction name="Buyer updates the Quote - in effect requesting a new price"  operation="quoteUpdate"
   channelVariable="Buyer2SellerC">
          <description type="documentation">Quot Update</description>
          <participate relationshipType="BuyerSeller"  fromRole="BuyerRoleType" toRole="SellerRoleType" />
                 <exchange name="updateQuote"  informationType="QuoteUpdateType" action="request">
          </exchange>
                <exchange name="acceptUpdatedQuote"  informationType="QuoteAcceptType" action="respond">
             <description type="documentation">Accept Updated Quote</description>
          </exchange>
        </interaction>
     </sequence>
   </choice>
 </workunit>
```

- **Bartering process**
  - **Set out "bateringDone" variable to "true"**
  - **Buyer updates the quote and gets a response back from the Seller**

WS-CDL Tool – Pi4SOA

- WS-CDL editor Pi4SOA
  - www.pi4tech.com
    - Plug-in Eclipse
    - Distributed by source forge with Apache 2.0 licence
  - Tree based editor based on structural clarity (see workunit explanation)
  - Testing a choreography by simulating messages that make up interactions.
  - Testing correct set of messages
    - Incorrect set of messages - results in a “SEVERE” error warning
  - Generate the code skeleton
    - WS-CDL to Java or WS-BPEL

Thank you!