ORQUESTRAÇÃO E COREOGRAFIAS DE SERVIÇOS WEB

Emílio Francesquini e Fabio Kon
IME - USP
{emilio, fabio.kon}@ime.usp.br

Congresso Brasileiro de Software
Salvador, 27 de setembro de 2010

Segunda-feira, logo pela manhã...

Marzullo, 2009

SERVIÇOS - INTUIÇÃO

"Um serviço é um tipo de relacionamento (contrato) entre um provedor e um consumidor, sendo que esse provedor se compromete em realizar determinadas tarefas com resultados pré-estabelecidos para um consumidor, e que, por sua vez, se compromete a usar o serviço da forma contratada."

Zhang, 2007

SERVIÇOS - DEFINIÇÃO

- O W3C define um web service como:

  "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.''

NA WEB NÃO FALTAM EXEMPLOS...

- RSS
- Google Calendar (CalDAV sobre HTTP) sincronizável com o iCal, por exemplo
- Amazon Web Services (SOAP ou REST sobre HTTP)
  - EC2
  - S3
  - SimpleDB
  - ...

ROTEIRO

- Serviços Web - Conceitos gerais
- XML/RPC e SOAP
- REST
- Orquestração
- Coreografias
- Desafios de Pesquisa

PRECURSORES

- RPC
- CORBA
- DCOM
- Java RMI

WEB SERVICES

- Um dos usos mais comuns de Web Services tem sido para a integração de sistemas "substituindo"
  - Troca de arquivos (FTP, diretório compartilhado, ...)
  - Tabelas em BD
  - TCP/IP
  - CORBA, JRMI, RPC...

WEB SERVICES - VANTAGENS

- Interface simples de ser implementada
- Qualquer linguagem que tenha possibilidade de utilizar soquetes TCP já está virtualmente apta a utilizar um web service
- Por ser sobre HTTP é mais provável que seja capaz de passar por firewalls, proxies, ...
- É o padrão de fato do mercado
- Baixo acoplamento e possibilidade de evolução da interface mantendo compatibilidade com versões anteriores

WEB SERVICES - DESVANTAGENS

- Desempenho
- Há uma grande disparidade entre o padrão especificado e do padrão "de fato"
  - Qualquer implementação que faça algo a mais que do que as tarefas básicas acaba oferecendo algumas surpresas no momento da integração com outros sistemas

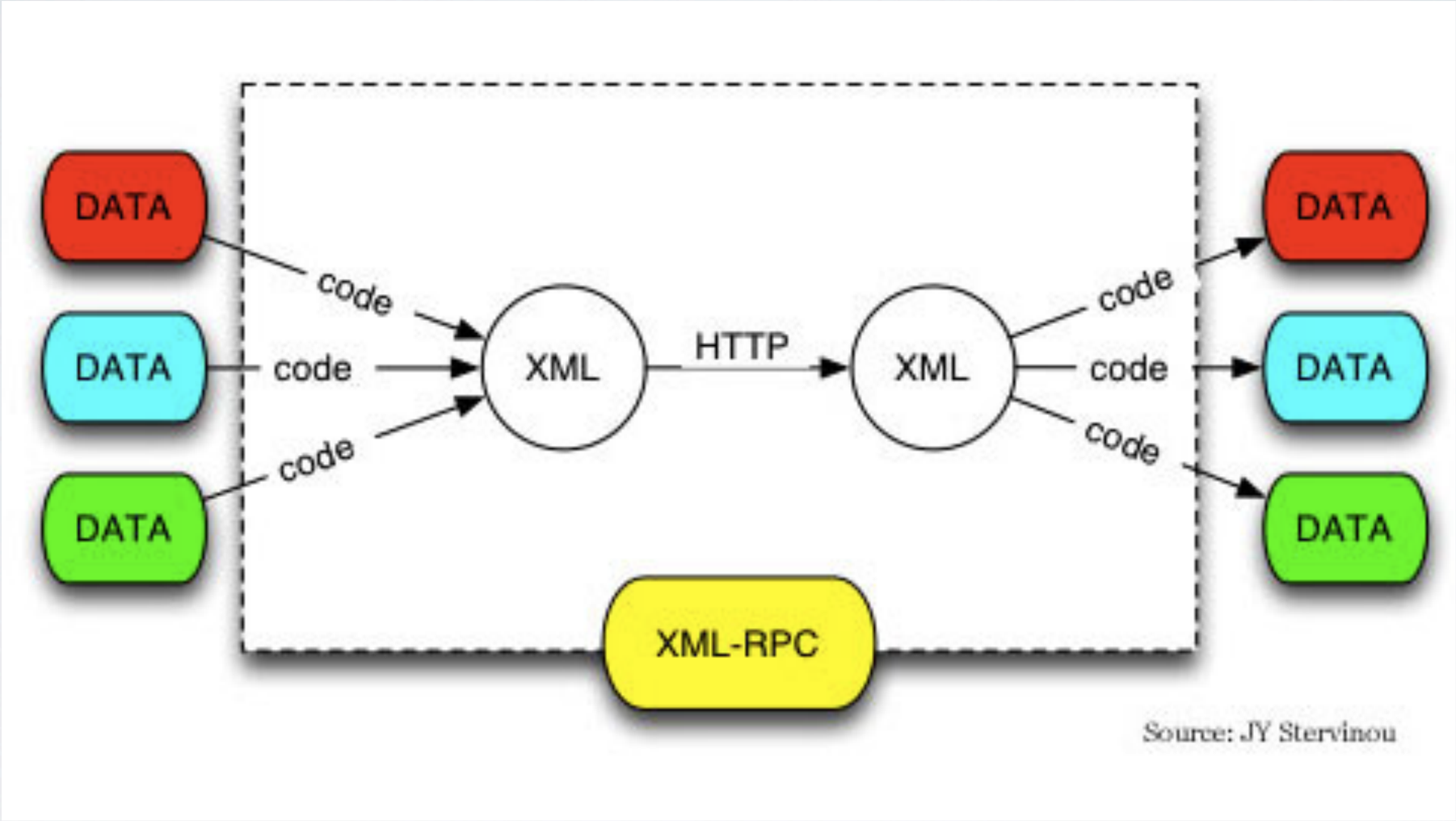
WEB SERVICES

- As maneiras mais comuns para fazer acesso aos web services são:
  - XML/RPC
  - SOAP (Simple Object Access Protocol)
  - REST (REpresentational State Transfer), também chamados de RESTful web services
- A comunicação, na maior parte dos casos, é feita utilizando XML sobre HTTP

XML/RPC

- É uma forma de se fazer RPC (chamadas remotas de procedimentos) através da Internet
- Usa
  - HTTP como protocolo de transporte
  - XML como linguagem de codificação (*marshalling*)
- 1ª implementação:
  - UserLand Software, 1998.
  - Dave Winer
- Serviu de inspiração para o protocolo SOAP que teve apoio da Microsoft

IDEIA DO XML/RPC



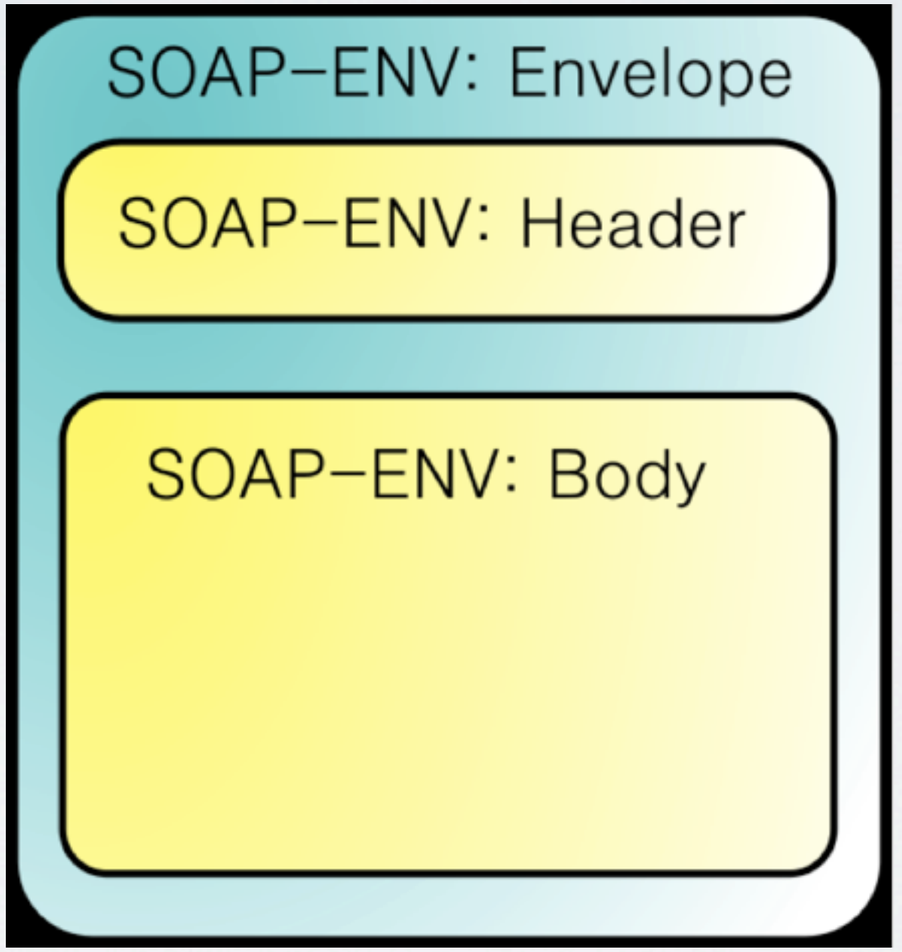
SOAP

- Protocolo padrão para interoperabilidade de sistemas heterogêneos.
- Criado por Dave Winer da UserLand Software em colaboração com Don Box, Bob Atkinson, and Mohsen Al-Ghosein na Microsoft.
- Desde 2003, é um protocolo padrão da W3C.
- No início, a sigla significava Simple Object Access Protocol
- Depois, decidiu-se que a sigla não significa nada :-)

A ESPECIFICAÇÃO DE SOAP

- Define 4 elementos:
  - O modelo de processamento - regras para processamento das mensagens
  - Modelo de extensibilidade - conceitos de funcionalidades extras (segurança, transações,...) e módulos SOAP
  - Protocolo de transporte - regras exigidas do protocolo usado para trocar mensagens SOAP (normalmente HTTP mas também SMTP, HTTPS, etc.)
  - A estrutura das mensagens SOAP

FORMATO DAS MENSAGENS



WSDL

- As interfaces em SOAP são definidas através da WSDL – Web Service Definition Language
- WSDL é um documento XML que traz operações, parâmetros, retornos, exceções e pontos de acesso

ELEMENTOS DA WSDL

- Types
  - Definições de tipos de dados
  - Descreve as mensagens trocadas
  - Utiliza o formato padrão W3C XML Schema
- Messages
  - Definições tipadas dos dados sendo trocados
- Port type
  - Coleção de operações
  - Definição de um serviço

EXEMPLO DE WSDL - HELLO WORLD

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://hello/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://hello/"
   name="HelloWorldService">
   <types>
      <xsd:schema>
         <xsd:import namespace="http://hello/"
            schemaLocation="http://localhost:8080/hello?xsd=1"></xsd:import>
      </xsd:schema>
   </types>
   <message name="sayHelloTo">
      <part name="parameter" element="tns:sayHelloTo"></part>
   </message>
   <message name="sayHelloToResponse">
      <part name="parameter" element="tns:sayHelloToResponse"></part>
   </message>
   <portType name="HelloWorld">
      <operation name="sayHelloTo">
         <input message="tns:sayHelloTo"></input>
         <output message="tns:sayHelloToResponse"></output>
      </operation>
   </portType>
```

ELEMENTOS DA WSDL

- Binding
  - Protocolo de comunicação e formato dos casos (codificação) para um *port type* específico
    - Exemplos de protocolo: SOAP 1.1 sobre HTTP ou sobre SMTP
    - Exemplos de codificação: SOAP ou RDF
- Port
  - Define o endereço para comunicação (ex. URL para HTTP ou e-mail para SMTP)
- Service
  - Um conjunto de *ports* relacionados entre si

EXEMPLO DE WSDL - HELLO WORLD

```
<binding name="HelloWorldPortBinding" type="tns:HelloWorld">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document"></soap:binding>
   <operation name="sayHelloTo">
      <input>
         <soap:body use="literal"></soap:body>
      </input>
      <output>
         <soap:body use="literal"></soap:body>
      </output>
   </operation>
</binding>

<service name="HelloWorldService">
   <port name="HelloWorldPort" binding="tns:HelloWorldPortBinding">
      <soap:address location="http://localhost:8080/hello"></soap:address>
   </port>
</service>
</definitions>
```

EXEMPLO DE REQUISIÇÃO SOAP

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>RPC Corp. Ltd.</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

fonte: w3schools.com

EXEMPLO DE RESPOSTA SOAP

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

fonte: w3schools.com

PONTOS FORTES E FRACOS

✓ Baseado em protocolos e formatos largamente utilizados (XML, HTTP, etc.).

✓ Fácil integração com servidores Web.

✓ Usa portas normalmente abertas em firewalls (gambiarra que deu certo).

x Mensagens grandes demais

- x Processamento muito lento no cliente e no servidor
- x Largura de banda usada muito grande
- x Extremamente ineficiente se comparado a protocolos binários como CORBA

EXEMPLO EM JAVA (JAX-WS)

```
   @WebService
public class HelloWorld {

   @WebMethod
   public String sayHelloTo(String name) {
      return "Hello " + name;
   }

   public static void main(String[] args) {
      HelloWorld service = new HelloWorld();
      Endpoint endpoint = Endpoint.create(service);
      endpoint.publish("http://localhost:8080/hello");
   }

}
```

EXEMPLO DE CLIENTE EM ERLANG

```
1> inets:start().
ok
2> Wsdl = yaws_soap_lib:initModel("http://localhost:8080/hello?wsdl").
{wsdl,[{operation,"HelloWorldService","HelloWorldPort",
                  "sayHelloTo","HelloWorldPortBinding",
                  "http://localhost:8080/hello",[]}],
...
3> yaws_soap_lib:call(Wsdl, "sayHelloTo", ["Machado de Assis"]).
{ok,undefined,
    [{'p:sayHelloToResponse',[],"Hello Machado de Assis"}]}
4>
```

REST

- *REpresentational State Transfer*
- Serviços que seguem essa linha são comumente chamados de *RESTful*
- Modelo apresentado na tese de doutoramento de Roy T. Fielding (UC Irvine, 2000)

REST - PRINCÍPIOS

- Atribuição de identificadores
- Associação de recursos
- Utilização de métodos padrão
- Multiplicidade de representações
- Comunicação sem manutenção de estado

REST - IDENTIFICADORES

- Todas as abstrações do sistema merecem ser identificadas
- Não apenas uma abstração individualmente deve ser identificada, mas qualquer conjunto destas abstrações que faça sentido também pode ser identificado por um id. Por exemplo:
  - Todas as latas de ervilha
  - Todas as pessoas que tem endereço em SP

REST - IDENTIFICADORES

- Na web, o jeito mais natural de fazer tal identificação é através de URIs
  - http://example.com/customers/1234
  - http://example.com/orders/2007/10/776654
  - http://example.com/orders/2007/11
  - http://example.com/products?color=green
- Sem paúra!
  - Anos de prática OO nos ensinaram a não expor os detalhes de implementação, ainda mais ids do BD
  - A idéia toda é a de que os conceitos (recursos) que você desejará expor são geralmente muito mais abrangentes do que uma linha no BD

REST - ASSOCIAÇÃO - HATEOAS

- HATEOAS - Hypermedia As The Engine Of Application State
  - Significa que tudo deve ser ligado. Considere o seguinte exemplo de uma resposta a uma requisição de pesquisa por pedidos

```
<pedido ref='http://xyz.com/pedido/1234'>
  <qtd>23</qtd>
  <produto ref='http://abc.com/produtos/4554'/>
  <cliente ref='http://jkl.com/clientes/7654'/>
</pedido>
```

REST - MÉTODOS PADRÃO

- Pode-se imaginar que todo recurso é algo semelhante a

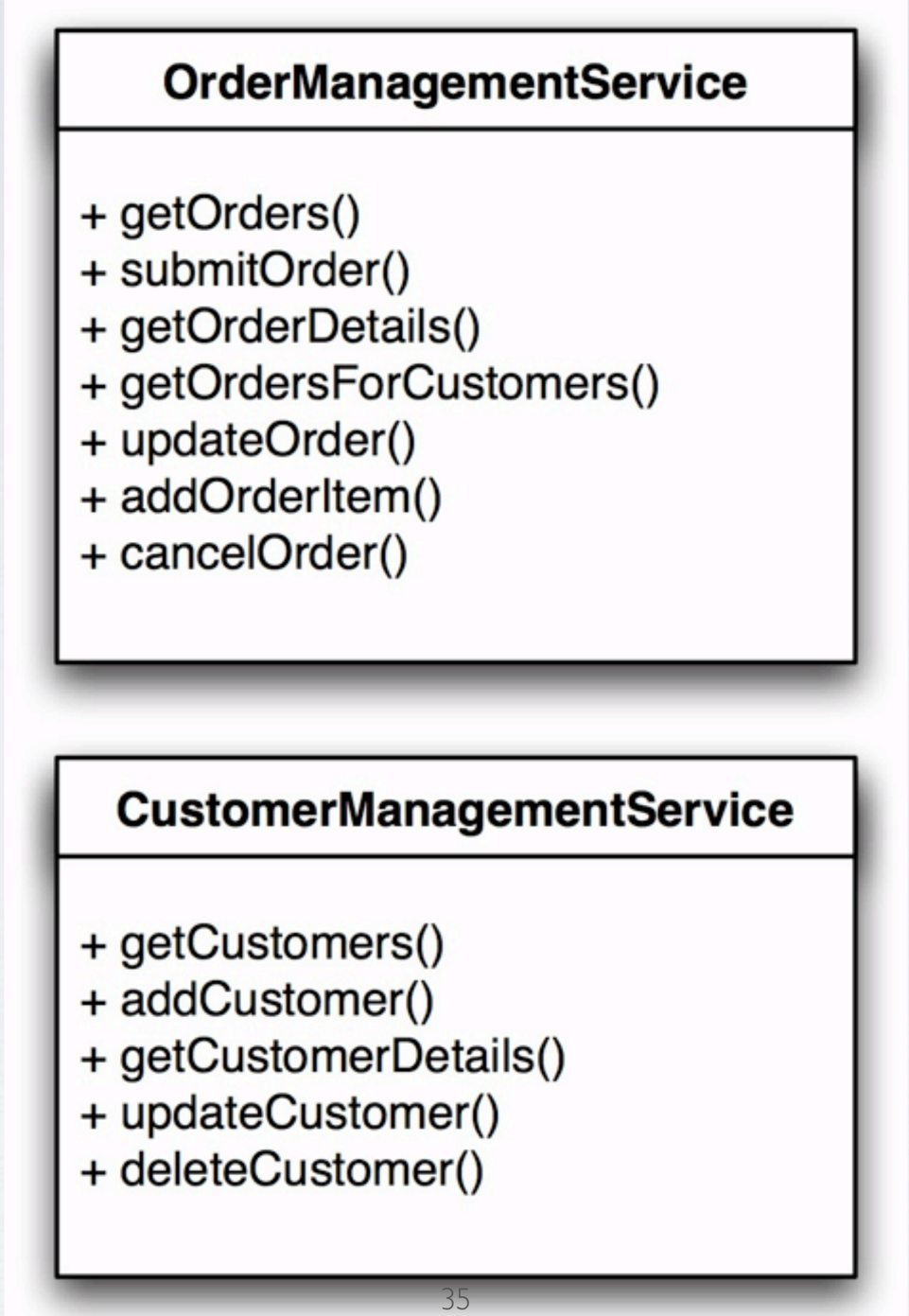
```
class Resource {
      Resource(URI u);
      Response get();
      Response post(Request r);
      Response put(Request r);
      Response delete();
}
```

REST - MÉTODOS PADRÃO

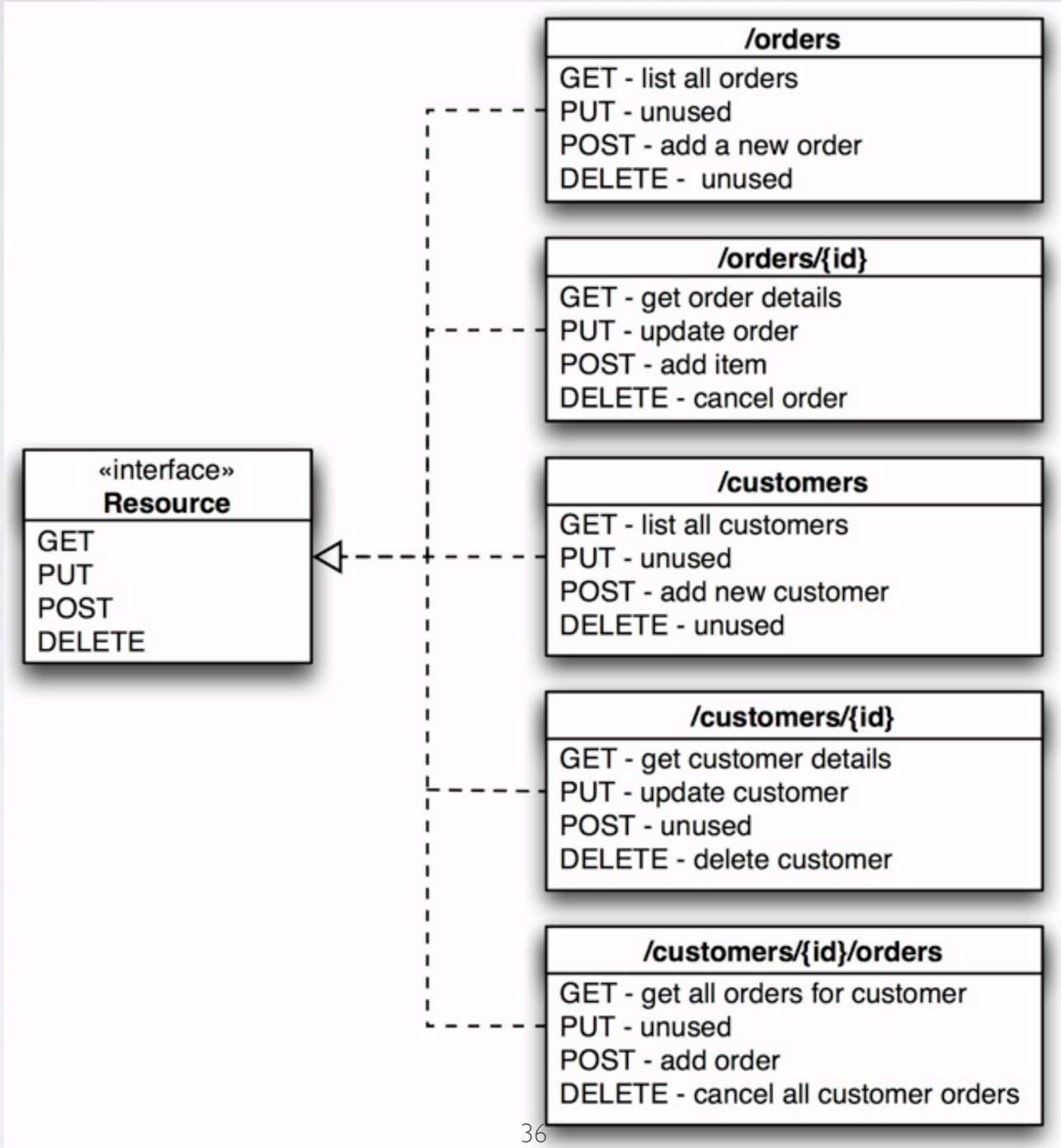
- GET – Pega uma representação do recurso
- PUT – Cria ou atualiza, se já existir o recurso, com as informações passadas
- DELETE – Apaga o recurso
- POST – Única operação não idempotente. Geralmente significa criação de um recurso ou ativação de algum processamento arbitrário, logo também não é seguro

REST - MÉTODOS PADRÃO

- Essas restrições devem ser respeitadas pelos serviços RESTful
  - No entanto, na vida real, nem todas essas restrições semânticas são respeitadas e é preciso avaliar cada caso de uso individualmente
- Pode ser difícil imaginar que uma aplicação modelada de uma maneira OO possa ser mapeada desta maneira


OrderManagementService
+ getOrders()
+ submitOrder()
+ getOrderDetails()
+ getOrdersForCustomers()
+ updateOrder()
+ addOrderItem()
+ cancelOrder()
CustomerManagementService
+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()

Tilkov, 2007


«interface»
Resource
GET
PUT
POST
DELETE
/orders
GET - list all orders
PUT - unused
POST - add a new order
DELETE - unused
/orders/{id}
GET - get order details
PUT - update order
POST - add item
DELETE - cancel order
/customers
GET - list all customers
PUT - unused
POST - add new customer
DELETE - unused
/customers/{id}
GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer
/customers/{id}/orders
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders

Tilkov, 2007

REST - MÚLTIPLAS REPRESENTAÇÕES

```
  GET /clientes/1234 HTTP/1.1
Host: xyz.com
Accept: application/meucliente+xml

GET /clientes/1234 HTTP/1.1
Host: xyz.com
Accept: text/x-vcard
```

- Possível uso: uma mesma API que gera as informações em HTML para uma página ou em um formato XML para ser usada por alguma aplicação

REST – COMUNICAÇÃO SEM MANUTENÇÃO DE ESTADO

- Depois de ver os princípios anteriores, fica claro como fazer isso
  - Atenção: Colocar na URI um ID de sessão não transforma seu web service em um RESTful
- Vantagens
  - Diminuição da carga do servidor
  - Escalabilidade
  - Independência da implementação do servidor
  - Independência entre servidores

HTTP RESTFUL

- Também tem sido chamado de WebAPI
- Muito usado para fazer *mashups*
- Um dos pilares para as páginas de conteúdo mais modernas
- Faz a junção das informações direto no cliente e não mais no servidor

REST

- A especificação propriamente dita não diz nada quanto a HTTP
- Também não especifica o número de operações, apenas determina que a interface deve ser padrão
- Não é demais supor que REST e HTTP tenham se influenciado mutuamente já que o Fielding, criador e descritor do REST também tenha definido muitos dos padrões HTTP

REST - EXEMPLOS

- Twitter
- Flickr
- Google Search
- Yahoo!
  - Geocoding
  - Maps
  - Music
  - ...

REST - IMPLEMENTAÇÃO EM JAVA
JAX-RS

- Criado para facilitar a implementação de serviços REST em Java
- Baseado em anotações
- Evita a tarefa um tanto entediante de fazer o processamento de URIs, XMLs, ...

REST - JAX-RS - SERVIÇO SIMPLES

```
@Path("/orders")
public class OrderEntryService {
    @GET
    public String getOrders() {...}
}
```

- http://exemplo.com/orders

Fonte: Burke 2008

REST - JAX-RS - LENDO OS PARÂMETROS

```
@Path("/orders")
public class OrderEntryService {
    @GET
    public String getOrders(@QueryParam("size")
                            @DefaultValue("50") int size)
    {
       ... method body ...
    }
}
```

- http:/exemplo.com/orders?size=50

Fonte: Burke 2008

REST - JAX-RS - COMPOSIÇÃO DE ENDEREÇOS

```
@Path("/orders")
public class OrderEntryService {
    @GET
    @Path("/{id}")
    public String getOrder(@PathParam("id") int orderId) {
       ... method body ...
    }
}
```

- **@Path**
- http://exemplo.com/orders/3333 mas não http://exemplo.com/orders/3333/entries
  - Possibilidade de se usar expressões regulares `@Path("{id: \\d+}")`

Fonte: Burke 2008

REST - JAX-RS - CONTENT-TYPE

```
@Path("/orders")
public class OrderEntryService {
    @GET
    @Path("{id}")
    @Produces("application/xml")
    public String getOrder(@PathParm("id") int orderId)
    {
        ...
    }
}
```

- Só será chamado caso o cliente envie no pedido que ele aceita **`application/xml`**
- O Firefox, por exemplo, envia como tipos aceitos:
  - `Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8`

Fonte: Burke 2008

REST - JAX-RS - NEGOCIAÇÃO DE CONTEÚDO

```
@Path("/orders")
public class OrderEntryService {
    @GET
    @Path("{id}")
    @Produces("application/xml")
    public String getOrder(@PathParm("id") int orderId)  {...}

    @GET
    @Path("{id}")
    @Produces("text/html")
    public String getOrderHtml(@PathParm("id") int orderId) {...}

    @GET
    @Path("{id}")
    @Produces("application/json")
    public String getOrderJson(@PathParm("id") int orderId) {...}
}
```


Fonte: Burke 2008

REST - JAX-RS - CODIFICAÇÃO DE CONTEÚDO (MARSHALLING)

```java
@XmlRootElement(name="order")
public class Order {
    @XmlElement(name="id")
    int id;

    @XmlElement(name="customer-id")
    int customerId;

    @XmlElement("order-entries")

    List<OrderEntry> entries;
...
}
```

```java
@Path("/orders")
public class OrderEntryService {
    @GET
    @Path("{id}")
    @Produces("application/xml"
    public Order getOrder(@PathParm("id") int orderId) {...}
}
```

Fonte: Burke 2008

COMPOSIÇÃO DE SERVIÇOS

- Serviços bem construídos têm como principais características:
  - Baixo acoplamento
  - Reutilização
  - Interoperabilidade
  - Localizável (*Discoverability*)
- São conceitos inter-relacionados
  - com 1 ou 2 satisfeitos, os demais vem naturalmente

COMPOSIÇÃO NO MUNDO REAL

- Um caso de uso real tipicamente envolve
  - mais de um web service
  - ordem/interdependência entre as requisições aos web services
  - tomadas de decisões durante o fluxo de trabalho com base nas informações obtidas até então
  - lidar com indisponibilidade de serviços
  - controle de transações
  - ...

COMPOSIÇÃO - EXEMPLO

- Sistema de vendas
  - cadastro de clientes
  - controle de estoque
  - cobrança
- Utilizado nos computadores das lojas

SOLUÇÃO 1

- Escrevo código na minha linguagem de programação favorita e embarco junto com o cliente do meu sistema:
  - Acesso aos web services
  - Tratamento de exceções
- Defino o fluxo de trabalho

SOLUÇÃO 2

- Copio o código e implanto junto com o site
- Agora, site e cliente compartilham o código com o fluxo, regras, ...

PROBLEMAS

- Os web services possivelmente são implementados nas mais diversas linguagens mas o fluxo e as informações sobre a relação entre eles passaram a ser específicadas em uma só linguagem
- Fica a cargo do programador
  - Paralelismo
  - Transação
  - Controle de falhas (ex. indisponibilidade de serviços)
- Mudanças nos web services, por menores que sejam, implicam atualização dos clientes e implantação de nova versão do site
- Cliente/Site podem ter excesso de comunicação com o(s) servidor(es)
- Limitação de dispositivos/Replicação de código.

SOLUÇÃO 3

- Cria-se um novo web service
  - contém todo o fluxo e regras da composição dos serviços
  - controla transações paralelismo e tratamento de exceções
- Os clientes, os sites, os dispositivos, ... todos passam a utilizar este web service diretamente

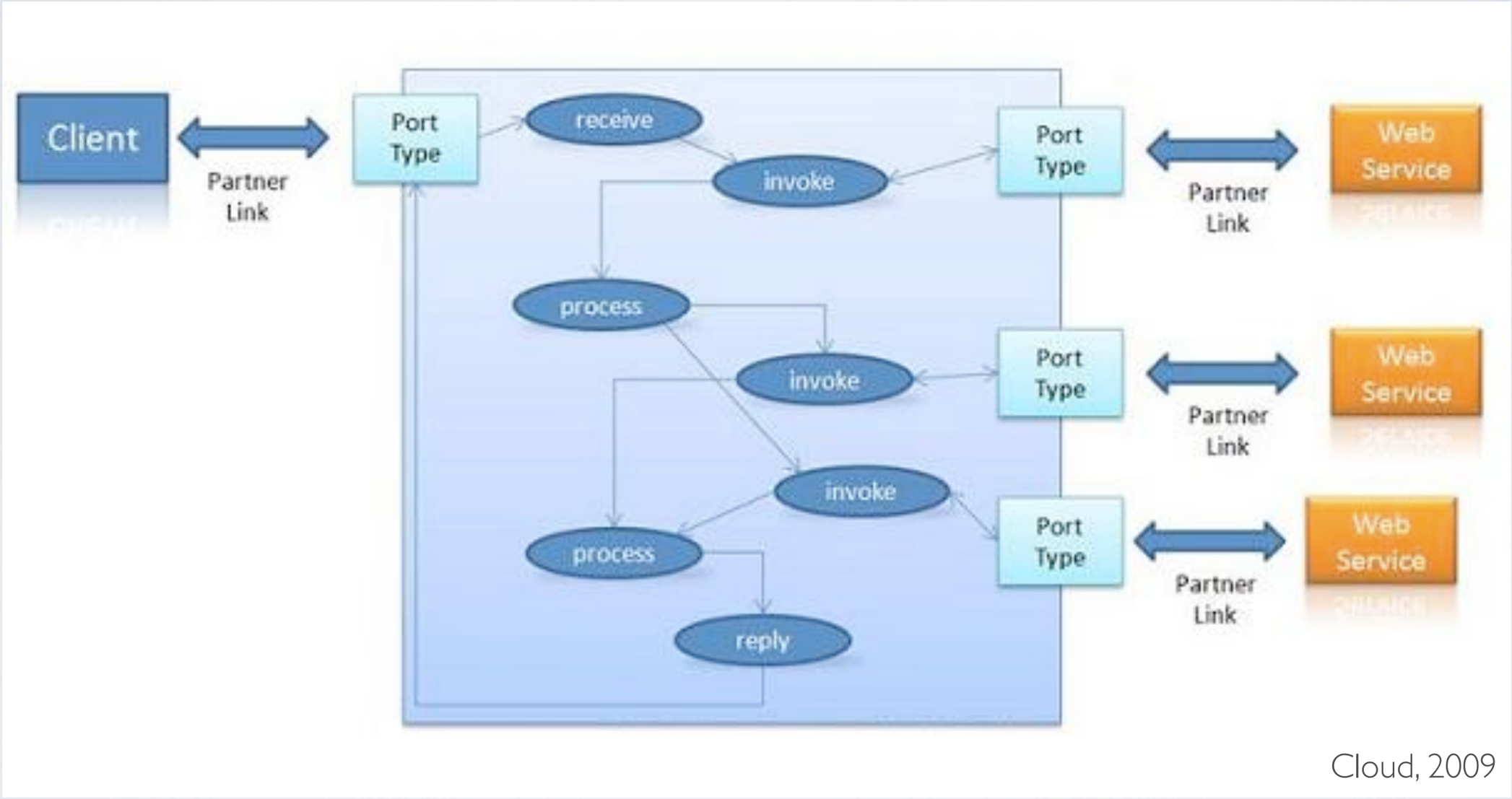
PORÉM, ALGUNS DOS PROBLEMAS PERMANECEM

- Fluxo vinculado a uma linguagem de programação específica
- Fica por conta do programador os controles de paralelismo, transações, falhas, ...
- A criação de uma nova composição exigiria ou
  - a replicação/utilização de uma biblioteca comum para o controle de transações, threads, ...
  - escrever mais código para a definição de relações entre os WS numa linguagem de programação específica

COMPOSIÇÃO DE SERVIÇOS WEB

- Orquestração
  - Centralizado
- Coreografias
  - Distribuído

DIAGRAMA DE COMPOSIÇÃO NO MUNDO REAL



Cloud, 2009

ORQUESTRAÇÃO

- BPEL - Business Process Execution Language
  - Surgiu em 2002 (BPEL4WS) num esforço conjunto de BEA (agora Oracle), IBM (WSFL) e Microsoft (XLANG). Nas versões seguintes juntaram-se SAP e Siebel Systems.
    - Em 2003 foi apresentado como um padrão aberto ao OASIS (WS-BPEL)
  - XML com extensão .bpel
  - Comumente chamado apenas de BPEL quando a versão específica não importa

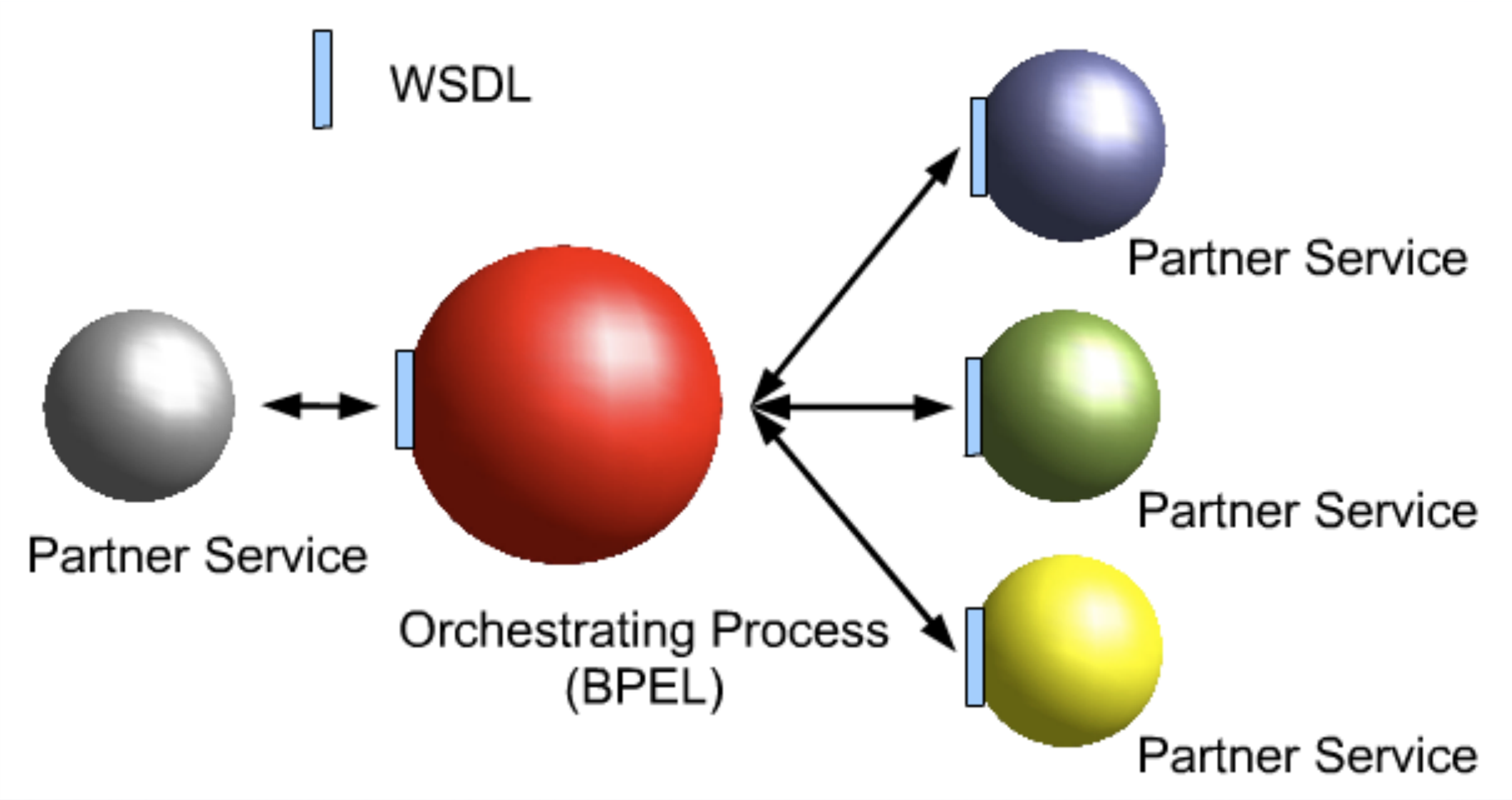
MOTIVAÇÕES PARA USAR BPEL

- Padrão de mercado
- Disponibilidade de ferramentas (pagas e gratuitas) de edição e mecanismos de execução de fluxos BPEL
  - Apache - ODE
  - Sun (agora Oracle) - OpenESB, NetBeans 6.X
  - Active Endpoints - The Active BPEL, ActiveBPEL Designer
  - Microsoft's BizTalk
  - Oracle - BPEL Process Manager
- Portabilidade entre diferentes mecanismos de execução*

DIGRESSÃO: BPMN

- BPMN - Business Process Management Notation
  - Padrão publicado pelo OMG (*Object Management Group*)
  - Permite analistas e arquitetos desenharem fluxos de negócio (alto nível)
- Dirigido às pessoas ligadas ao negócio enquanto BPEL é mais dirigido aos técnicos
- Servem normalmente como representações de como as coisas funcionam, no entanto, existem ferramentas que a partir de um documento BPMN são capazes de gerar um documento BPEL
  - A maneira pela qual esse mapeamento é feito está descrita no padrão BPMN

BPEL



Shin, 2010

ESTRUTURA DO ARQUIVO BPEL

```
<process ...>
  <!- Definições dos papéis dos processos participantes -->
  <partnerLinks>...</partnerLinks>
  <!- Variáveis utilizadas pelo processo -->
  <variables>...</variables>
  <!- Propriedades para permitir conversações -->
  <correlationSets>...</correlationSets>
  <!- Recuperação de erros -->
  <compensationHandlers>...</compensationHandlers>
  <!-Eventos concorrentes com o próprio processo -->
  <eventHandlers>...</eventHandlers>
  <!- Código do fluxo de negócio -->
  (atividades)
</process>
```

ATIVIDADES DA BPEL

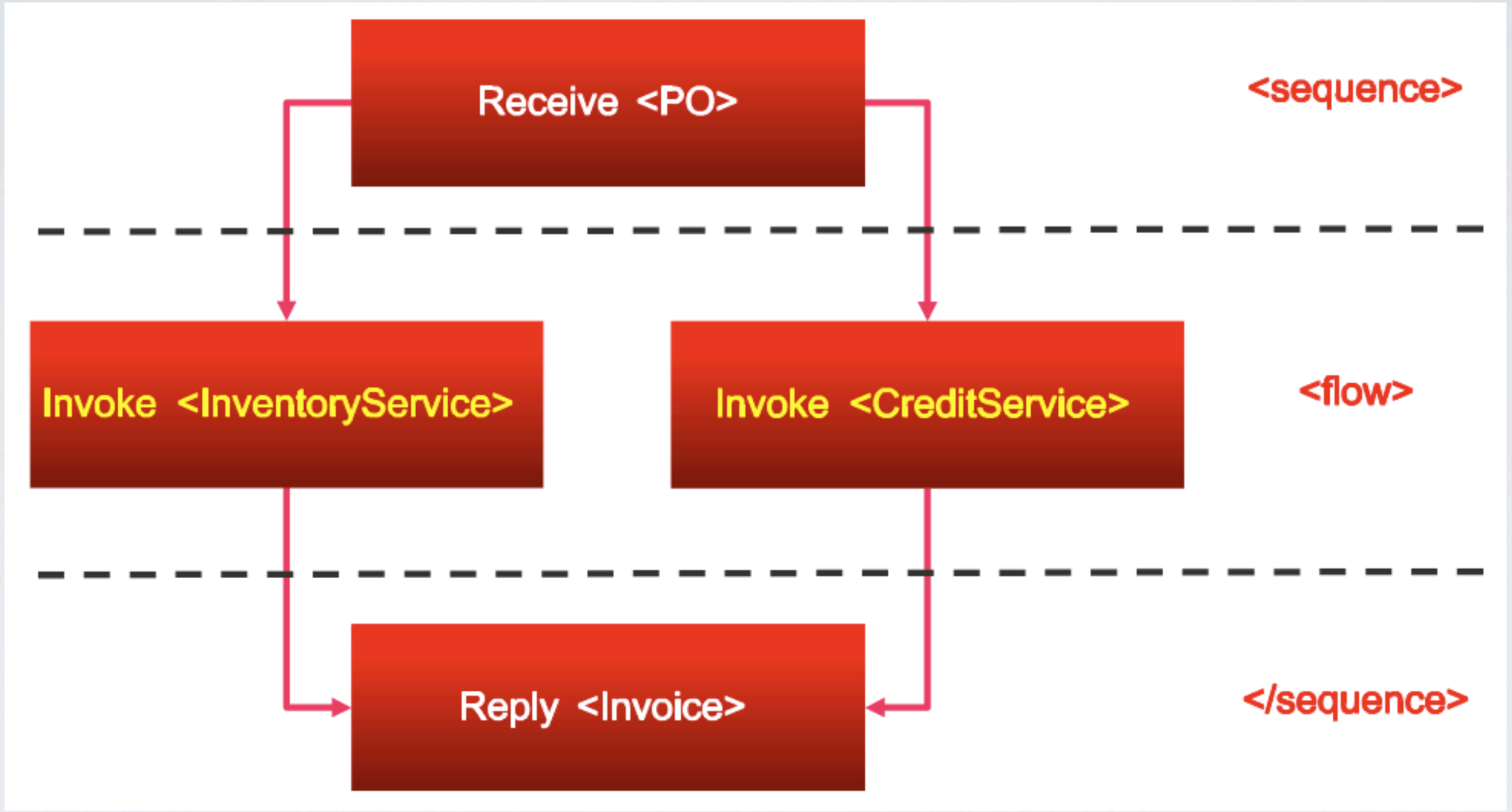
- Básicas
  - <invoke>
  - <receive>
  - <reply>
  - <assign>
  - <throw>
  - <wait>
  - <empty>
  - <exit>
- Estruturadas
  - <if>
  - <while>
  - <repeatUntil>
  - <foreach>
  - <pick>
  - <flow>
  - <sequence>
  - <scope>

BPEL - ATIVIDADES BÁSICAS

- `<invoke>`
  - Para invocar uma operação em um *portType* oferecido por um *partner*
- `<receive>`
  - Espera pelo recebimento de uma mensagem. Geralmente é o ponto de entrada para o fluxo
- `<reply>`
  - Utilizado para enviar uma mensagem como resposta ao recebimento de uma mensagem através do **`<receive>`**
  - A combinação **`<receive>/<reply>`** cria uma operação requisição/resposta no *portType* da WSDL do processo

BPEL - ATIVIDADES ESTRUTURADAS

- `<sequence>`
  - Executa as atividades contidas em ordem
- `<flow>`
  - Executa as atividades contidas em paralelo
- `<if>`
  - Execução condicional das atividades
- `<scope>`
  - Agrupa diversas atividades em um único escopo


Receive <PO>
<sequence>
Invoke <InventoryService>
Invoke <CreditService>
<flow>
Reply <Invoice>
</sequence>

Shin, 2010

```
<sequence>
 <receive partnerLink="customer" portType="lns:purchaseOrderPT"
          operation="sendPurchaseOrder" variable="PO"
          createInstance="yes" />
 <flow>
  <invoke partnerLink="inventoryChecker" portType="lns:inventoryPT"
          operation="checkINV" inputVariable="inventoryRequest"
          outputVariable="inventoryResponse" />

  <invoke partnerLink="creditChecker" portType="lns:creditPT"
          operation="checkCRED" inputVariable="creditRequest"
          outputVariable="creditResponse" />
 </flow>
 ...
 <reply partnerLink="customer" portType="lns:purchaseOrderPT"
        operation="sendPurchaseOrder" variable="invoice"/>
</sequence>
```

BPEL - EXEMPLO SÍNCRONO

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="SynchronousSample" ...>

  <import namespace="http://SynchronousSample"
          location="SynchronousSample.xsd"
          importType="http://www.w3.org/2001/XMLSchema"/>

  <import namespace="http://SynchronousSample"
          location="SynchronousSample.wsdl"
          importType="http://schemas.xmlsoap.org/wsdl/"/>
```

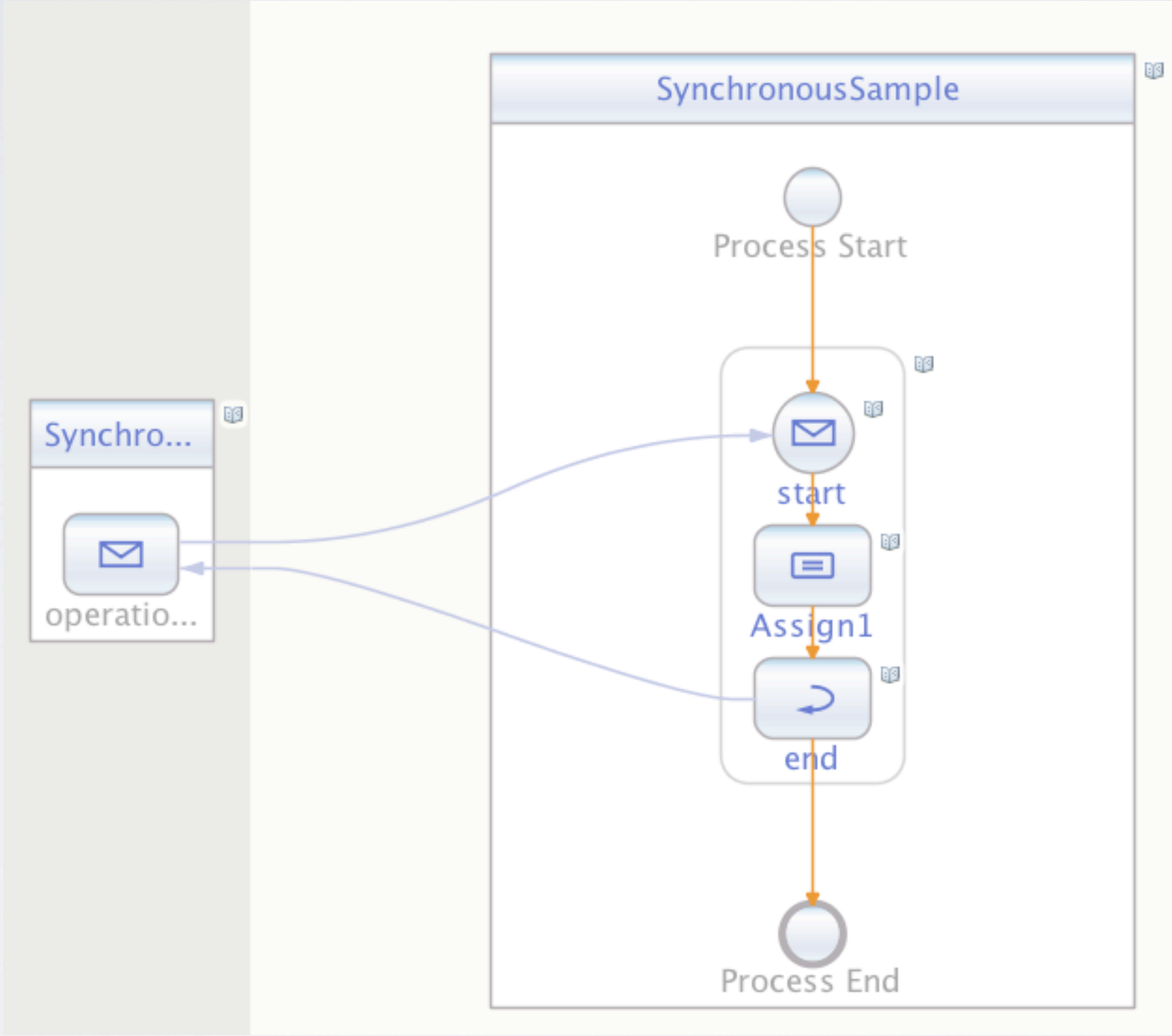
BPEL - EXEMPLO SÍNCRONO

```
  <partnerLinks>
  <partnerLink
    name="SynchronousSample"
    partnerLinkType="ns1:partnerlinktype1"
    myRole="partnerlinktyperole1">
  </partnerLink>
</partnerLinks>

<variables>
  <variable name="outputVar"
            messageType="ns1:responseMessage"/>
  <variable name="inputVar"
            messageType="ns1:requestMessage"/>
</variables>
```

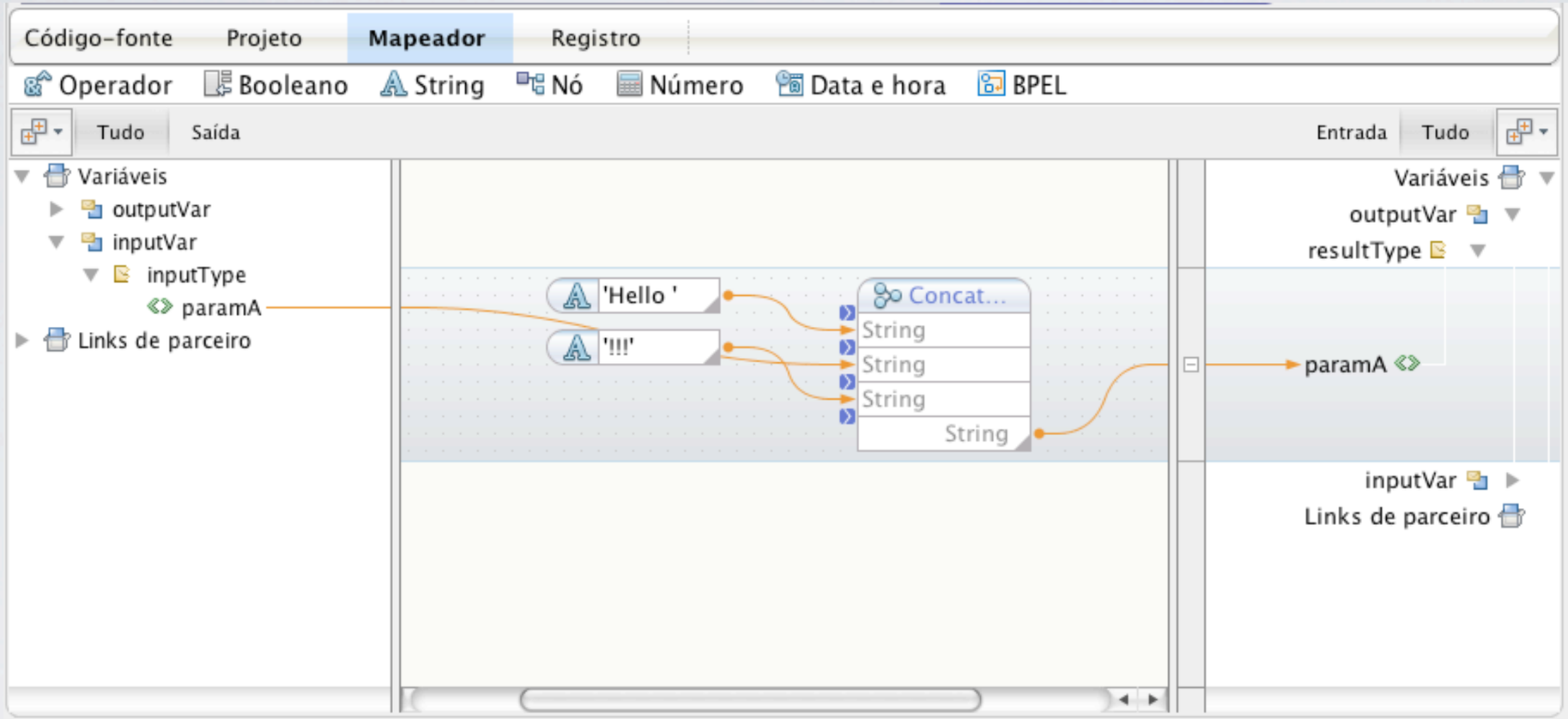
BPEL - EXEMPLO SÍNCRONO

```
<sequence>

<receive name="start" partnerLink="SynchronousSample" operation="operation1"
         portType="ns1:portType1" variable="inputVar" createInstance="yes"/>

<assign name="Assign1">
  <copy>
    <from>concat('Hello ', $inputVar.inputType/ns2:paramA, '!!!')</from>
    <to>$outputVar.resultType/ns2:paramA</to>
  </copy>
</assign>

<reply name="end" partnerLink="SynchronousSample" operation="operation1"
       portType="ns1:portType1" variable="outputVar">
</reply>

</sequence>
</process>
```

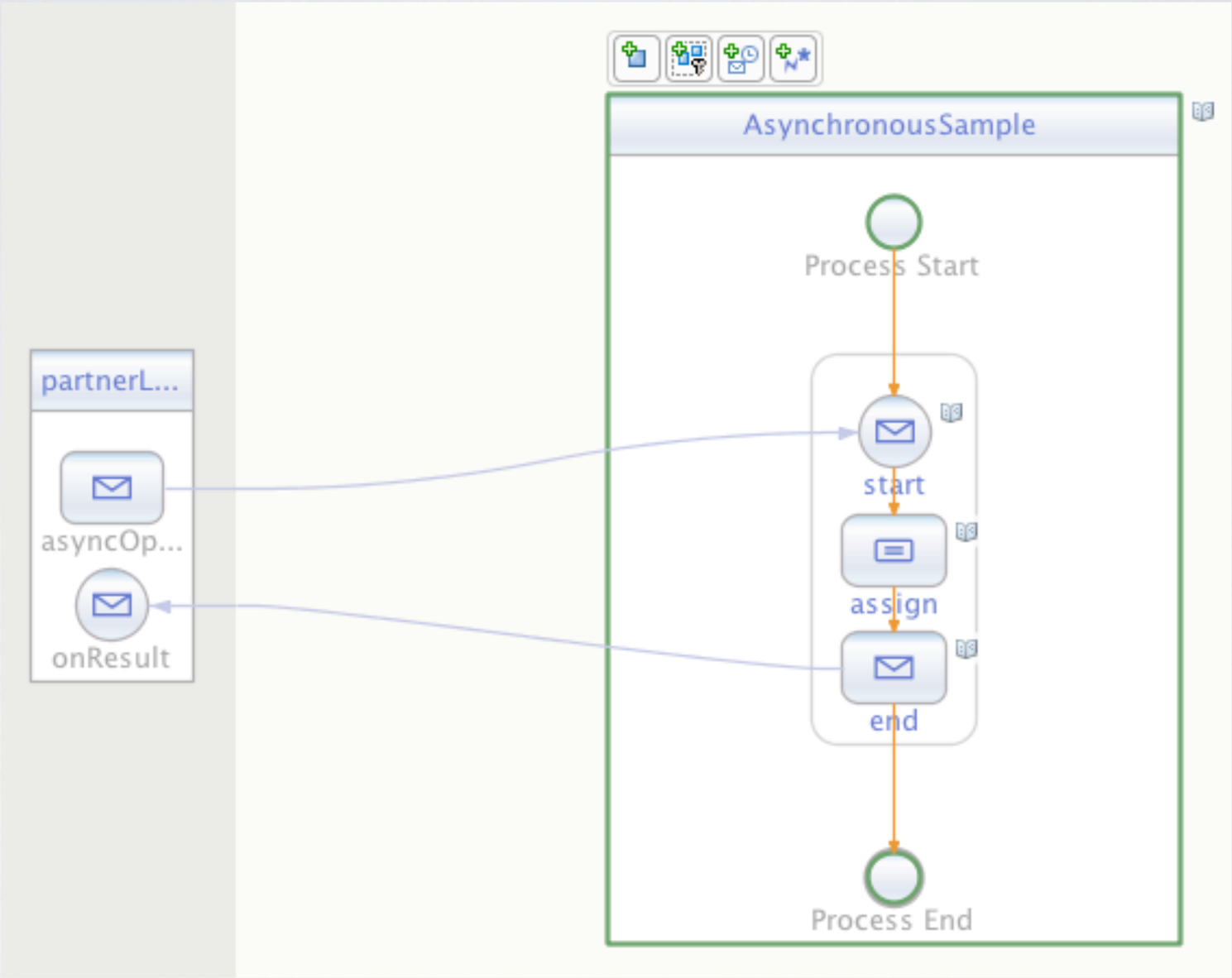
BPEL - EXEMPLO SÍNCRONO



BPEL - EXEMPLO SÍNCRONO



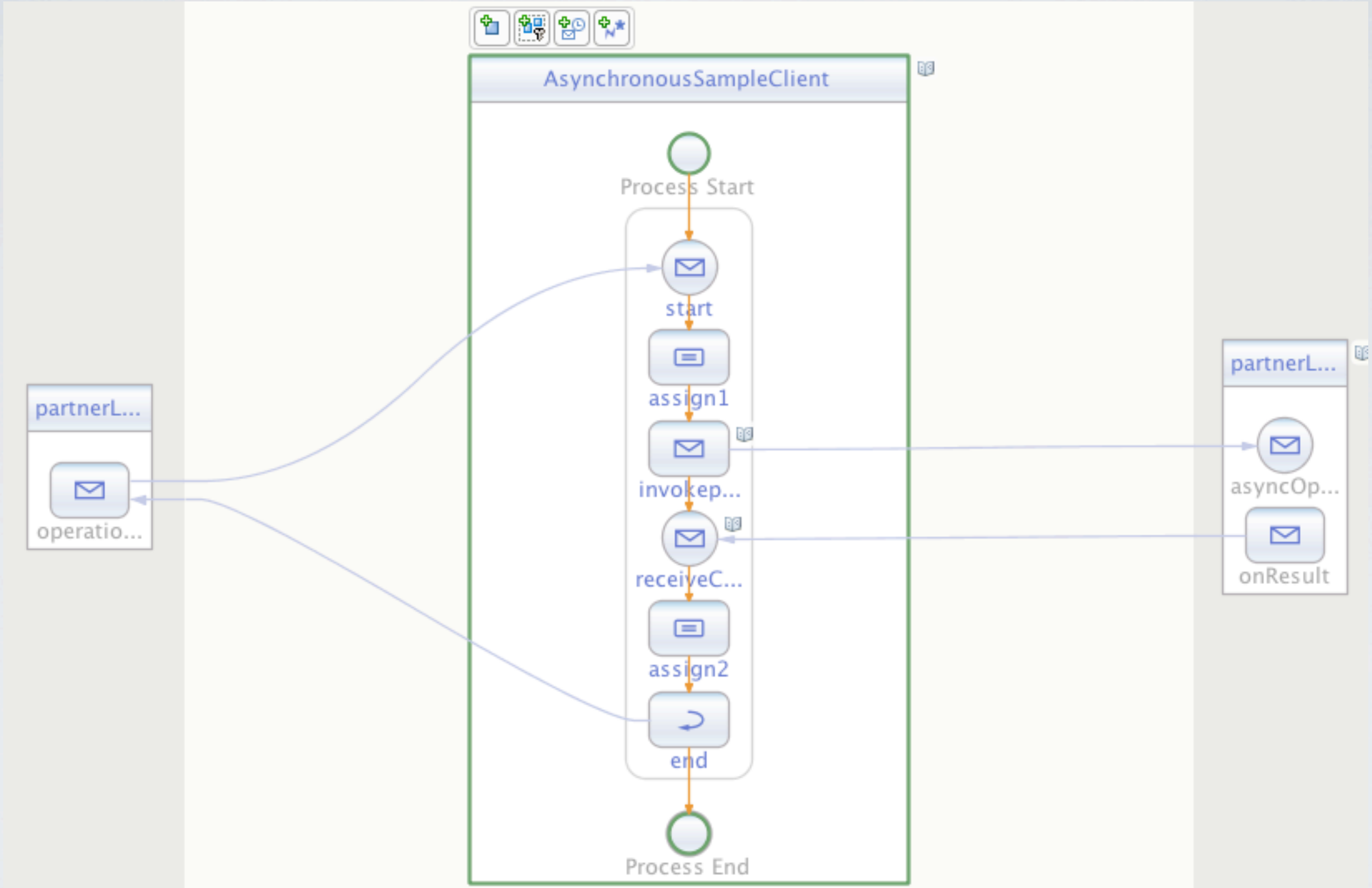
BPEL - EXEMPLO ASSÍNCRONO



BPEL - EXEMPLO ASSÍNCRONO

```
<sequence>
<receive name="start" partnerLink="partnerLinkA”
         portType="wsdlNS:MyPortType"
         operation="asyncOperation" variable="inputVar"
         createInstance="yes"/>

<assign name="assign">
  <copy>
    <from variable="inputVar" part="inputType"/>
    <to variable="outputVar" part="resultType"/>
  </copy>
</assign>

<invoke name="end" partnerLink="partnerLinkA”
        portType="wsdlNS:MyCallbackPortType"
        operation="onResult" inputVariable="outputVar"/>
</sequence>
```

BPEL - FAZENDO ACESSO A UM SERVIÇO ASSÍNCRONO



BPEL ASSÍNCRONO - PARTE I

```
<partnerLinks>
  <partnerLink
       name="partnerLinkA"
       partnerLinkType="wsdlNS:AsynchronousClientPartnerLinkType"
       myRole="AsynchronousClientProvider"/>

  <partnerLink
       name="partnerLinkB"
       partnerLinkType="partnerNS:AsyncPartnerLinkType"
       myRole="serviceRequestor"
       partnerRole="serviceProvider">

   </partnerLink>
</partnerLinks>
<correlationSets>
    <correlationSet name="correlator" properties="wsdlNS:correlatorProp"/>
</correlationSets>
```

BPEL ASSÍNCRONO - PARTE 2

```
<sequence>
<receive name="start" partnerLink="partnerLinkA"
         portType="wsdlNS:MyPortTypeClient"
         operation="operationA" variable="inputVar" createInstance="yes">
  <correlations>
    <correlation set="correlator" initiate="yes"/>
  </correlations>
</receive>

<assign name="assign1">
  <copy>
    <from variable="inputVar" part="inputType"/>
     <to variable="partnerInputVar" part="inputType"/>
  </copy>
</assign>

<invoke name="invokepartner" partnerLink="partnerLinkB"
        portType="partnerNS:MyPortType"
        operation="asyncOperation" inputVariable="partnerInputVar"/>
```

BPEL ASSÍNCRONO - PARTE 3

```
  <receive name="receiveCallback" partnerLink="partnerLinkB"
           portType="partnerNS:MyCallbackPortType"
           operation="onResult" variable="partnerOutputVar" createInstance="no">
    <correlations>
      <correlation set="correlator"/>
    </correlations>
  </receive>

  <assign name="assign2">
    <copy>
      <from variable="partnerOutputVar" part="resultType"/>
      <to variable="outputVar" part="resultType"/>
    </copy>
  </assign>

  <reply name="end" partnerLink="partnerLinkA"
         portType="wsdlNS:MyPortTypeClient"
         operation="operationA" variable="outputVar"/>
</sequence>
```

COREOGRAFIAS

- Coreografias de Serviços Web é uma técnica para composição de serviços de forma distribuída e descentralizada, vista sob uma perspectiva global.
- Não há um nó coordenador.
- Cada nó sabe o que deve fazer; Cada nó sabe como colaborar com seus vizinhos na coreografia.
- A coreografia é o resultado do conhecimento e comportamento coletivo espalhado pelo sistema.

A METÁFORA

- Bailarinos dançam seguindo uma coreografia que descreve um cenário global, sem um ponto único de controle.
- Cada dançarino desempenha um papel.
- Esse papel define as suas ações e a sua interação com os demais bailarinos do espetáculo.

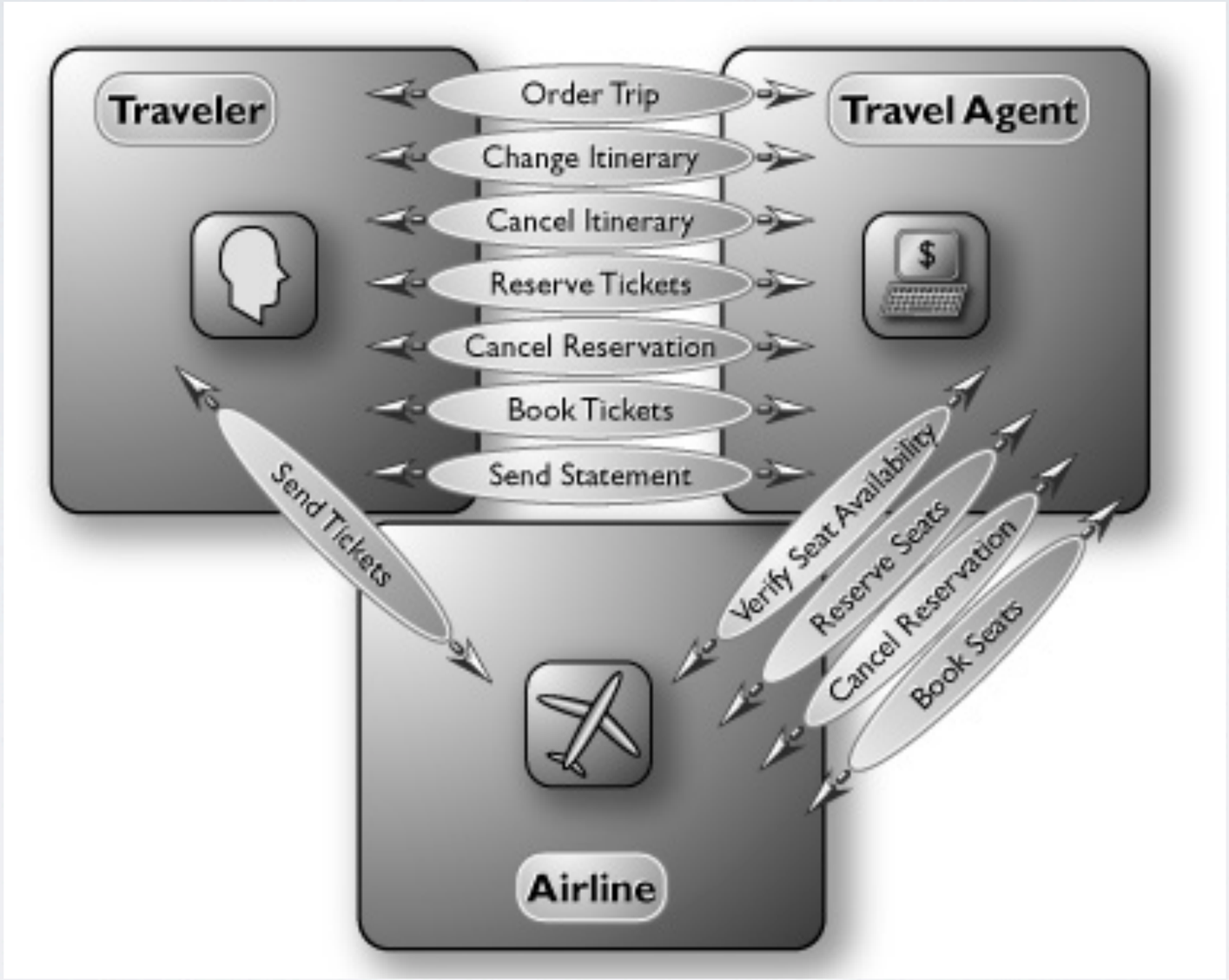
MOTIVAÇÃO

- Orquestrações possuem o controle centralizado
- O nó central é um ponto único de falha
- Arquiteturas centralizadas, em geral, tem problemas de
  - escalabilidade
  - conectividade (em redes de grande escala)
  - adaptabilidade
  - configurabilidade
- Coreografias tentam resolver essas limitações distribuindo as responsabilidades.

COMO FUNCIONA UMA COREOGRAFIA

- Cada nó/processo participante da coreografia sabe:
  - quando atuar
  - com quais outros processos deve interagir
  - quais operações deve executar
  - quais mensagens deve trocar
  - qual o momento adequado para essa troca de mensagens

EXEMPLO DE CENÁRIO DE USO



HISTÓRIA

- W3C Workshop on Web Services, 2001
- muitas apresentações apontaram a necessidade de padrões permitindo a composição de serviços de uma forma descentralizada, i.e., coreografias.
- W3C Web Services Architecture Working Group
- considera coreografias como um fator crítico para o sucesso para serviços nascentes da Web
- Criou-se então em 2003 o *W3C Architecture Domain Web Services Choreography* Working *Group Charter*

REQUISITOS PARA UMA LINGUAGEM PARA COREOGRAFIAS (W3C)

- Composições
- Associações
- Troca de Mensagens
- Gerenciamento de Estado
- Dois aspectos a serem considerados:
  - interface, descrição externa, interação com outros serviços
  - execução, descrição interna, comportamento do serviço

REQUISITOS PARA COMPOSIÇÃO

- Coreografia é um serviço web: modelo de composição recursivo.
- Definição do comportamento externo observável de uma coreografia.
- Habilidade de representar coreografias com estado.
- Identificação das instâncias das coreografias.
- Gerenciamento do ciclo de vida (criação, término, etc.).
- Interações entre serviços através de trocas de mensagens.
- Definição de comportamento, regras de escopo, atividades.

REQUISITOS PARA ASSOCIAÇÕES

- Papéis baseado no uso de serviços web.
- Ligações entre serviços web.
- Referências a serviços web.

REQUISITOS PARA TROCA DE MENSAGENS

- Conversas - trocas de mensagens inter-relacionadas que definem a interação entre serviços.
- Correlações e gerenciamento do seu ciclo de vida.
- Correlação com instâncias de coreografias e seus estados.

REQUISITOS PARA GERENCIAMENTO DE ESTADO

- Definição de estado
- Manipulação de estado
- Consulta ao estado

PRINCIPAIS PADRÕES PARA COREOGRAFIAS

- WSCI (W3C, 2002)
- WS-CDL (W3C, 2005)

- BPSS by ebXML
- WSFL by IBM
- XLANG by Microsoft
- BPEL4WS by IBM, Microsoft and BEA
- BPML, BPMN, BPMN2

WSCI
WEB SERVICE CHOREOGRAPHY INTERFACE

- É uma linguagem de definição de interfaces, baseada em XML (W3C, 2002).
- Descreve o fluxo de mensagens entre os participantes de uma coreografia.
- Trabalha em conjunto com WSDL mas descreve os aspectos dinâmicos da interface de um serviço em função de sua interface estática.
- pronúncia: whiskey (como a bebida)

WSCI DEFINE O COMPORTAMENTO OBSERVÁVEL DE UM SERVIÇO

- Comportamento é expresso em termos de
  - dependências lógicas e temporais entre as mensagens trocadas
  - regras de sequenciamento, correlações, tratamento de exceções e transações.
- WSCI descreve as interações coletivas entre os serviços Web, provendo uma visão global, orientada pelas mensagens das interações.

DESCRIÇÃO DAS INTERDEPENDÊNCIAS

- WCSI descreve as interdependências entre as operações de serviços Web de forma que qualquer cliente possa:
  - interagir com esse serviço no contexto de um determinado processo e
  - antecipar o comportamento esperado desse serviço em qualquer momento do seu ciclo de vida.

PERGUNTAS RESPONDIDAS POR UMA ESPECIFICAÇÃO WSCI

- Qual a sequência de mensagens esperada por um serviço?
- Quais regras são usadas para definir a sequência de mensagens?
- Qual a relação entre as mensagens de entrada e de saída de um serviço?
- Quando uma sequência de mensagens começa e quando termina?
- Uma sequência de mensagens pode ser desfeita?

VISÃO GLOBAL

- WCSI permite definir uma visão global da coreografia de modo que se possa
  - verificar se o serviço se comporta da forma como está especificado (V&V)
  - derivar a coreografia de todo o processo de forma a descrevê-la e modificá-la
  - monitorar o comportamento do serviço em relação a todos os participantes da troca de mensagens.

ELEMENTOS DE WSCI

- **Coreografia de mensagens**: ordem na qual mensagens são trocadas.
- **Transações**: define quais operações são executadas dentro de transações distribuídas da forma tudo-ou-nada.
- **Tratamento de exceções**: descreve padrões de comportamentos alternativos quando coisas dão errado.
- **Gerenciamento de Threads**: descreve se e como serviços tratam múltiplas conversas simultâneas.
- (continua)

ELEMENTOS DE WSCI

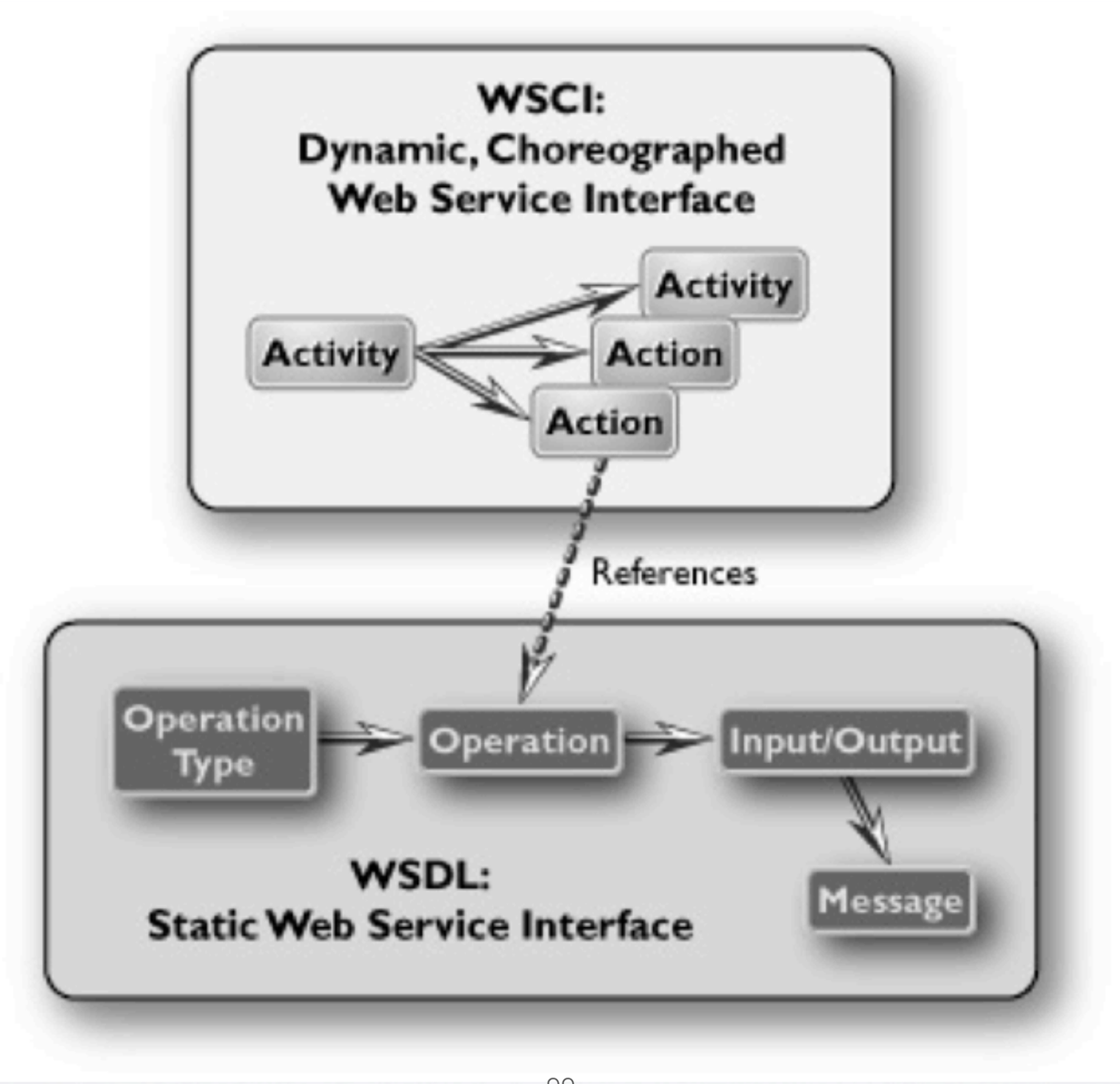
- **Propriedades e seletores**: descrevem os elementos que modificam o comportamento observável de um serviço, tais como variações baseadas em valores de partes de mensagens.

- **Conectores**: descrevem as ligações entre operações consumidoras de um serviço e operações produtoras de outro serviço.

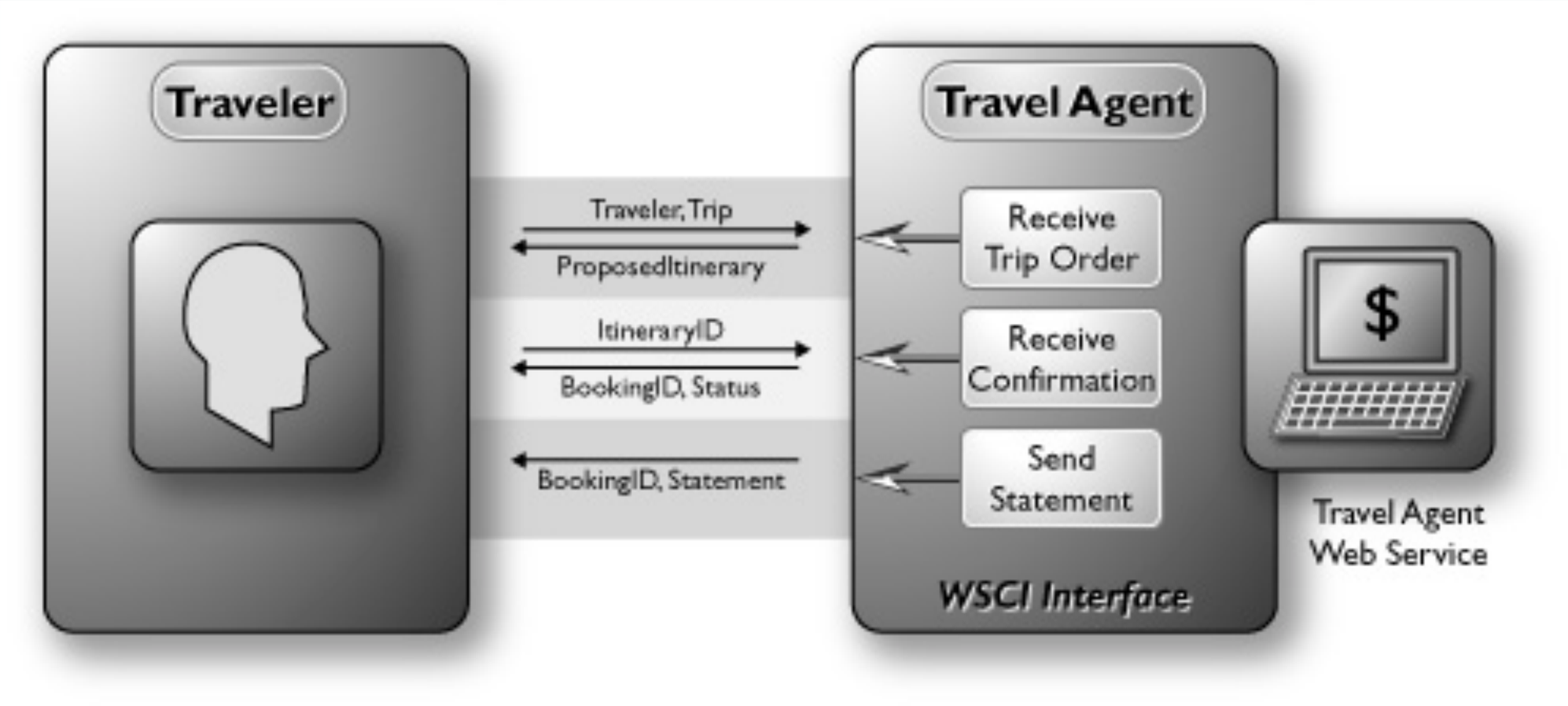
- **Contexto operacional**: descreve como um mesmo serviço se comporta no contexto de diferentes trocas de mensagens.

- **Participação dinâmica**: descreve como um serviço específico é selecionado em tempo de execução para compor uma coreografia.

RELAÇÃO COM WSDL



EXEMPLO



WSDL consegue definir as 3 operações isoladamente mas não descreve como se dá a sequência de mensagens, a coreografia em si.

O CÓDIGO WSCI (1/3)

```
<? xml version = "1.0" ?>
<wsdl:definitions name = "Travel Agent Dynamic Interface"
   targetNamespace = "http://example.com/consumer/TravelAgent"
   xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema"
   xmlns:tns = "http://example.com/consumer/TravelAgent"
   xmlns = "http://www.w3.org/2002/07/wsci10">

   <!-- WSDL complex types -->
   <!-- WSDL message definitions -->
   <!-- WSDL operations and port types -->
   <!-- selectors -->

   <correlation name = "itineraryCorrelation"
            property = "tns:itineraryID">
   </correlation>
```

mensagens com um mesmo itineraryID pertencem à mesma conversa

O CÓDIGO WSCI (2/3)

```
<interface name = "TravelAgent">
  <process name = "PlanAndBookTrip"
       instantiation = "message">

    <sequence>
    <action name = "ReceiveTripOrder"
        role = "tns:TravelAgent"
        operation = "tns:TAtoTraveler/OrderTrip">
    </action>

    <action name = "ReceiveConfirmation"
        role = "tns:TravelAgent"
        operation = "tns:TAtoTraveler/bookTickets">
       <correlate correlation="tns:itineraryCorrelation"/>

      <call process = "tns:BookSeats" />
    </action>

    <action name = "SendStatement"
        role = "tns:TravelAgent"
        operation = "tns:TAtoTraveler/SendStatement"/>
    </action>
    </sequence>
  </process>
```

processo se inicia quando a primeira mensagem é recebida

as 3 ações são executadas sequencialmente

o papel lógico do serviço nesse processo

a confirmação é relacionada a um pedido feito anteriormente

chamada a outro processo

O CÓDIGO WSCI (3/3)

```
    <process name = "BookSeats" instantiation = "other">
        <action name = "bookSeats"
            role = "tns:TravelAgent"
            operation = "tns:TAtoAirline/bookSeats">
        </action>
    </process>
  </interface>
</wsdl:definitions>
```

esse processo será disparado não pelo recebimento de uma mensagem mas sim por uma chamada explícita de um outro processo dentro da coreografia.

ATIVIDADES EM WSCI

- podem ser atômicas (operação definida em WSDL) ou compostas:
  - execução em sequência
  - execução em paralelo
  - laços (for-each, while, repeat-until)
  - condicional baseada na avaliação de uma condição (switch) ou na ocorrência de um evento (choice) que pode ser recebimento de mensagem associada a papel, ação ou mensagem, falha, timeout.

PROCESSOS EM WSCI

- Podem ser
  - disparados a partir do recebimento de uma mensagem específica,
  - criados a partir de uma operação do tipo **call** (a chamada fica bloqueada esperando pela conclusão da chamada) ou
  - criados em um novo thread a partir de uma operação do tipo **spawn**.

EXCEÇÕES EM WSCI

- É possível definir casos excepcionais e quais atividades devem ser executadas quando a exceção ocorre.
- Podem ser consideradas exceções:
  - mensagens específicas recebidas por um serviço
  - ocorrência de uma falha (tanto falha interna do serviço quanto o recebimento de uma mensagem de falha)
  - ocorrência de um *timeout*.

TRANSAÇÕES EM WSCI

- A coreografia pode definir que um conjunto de atividades deve ser executada de uma forma tudo-ou-nada. Ou seja
- Se nenhuma exceção ocorrer, a transação chega ao fim.
- Se uma exceção ocorrer, todas as atividades da transação são desfeitas e, opcionalmente, um conjunto de atividades de compensação é executado.
- Transações podem ser simples ou aninhadas.

APLICAÇÃO NA AIRLINE

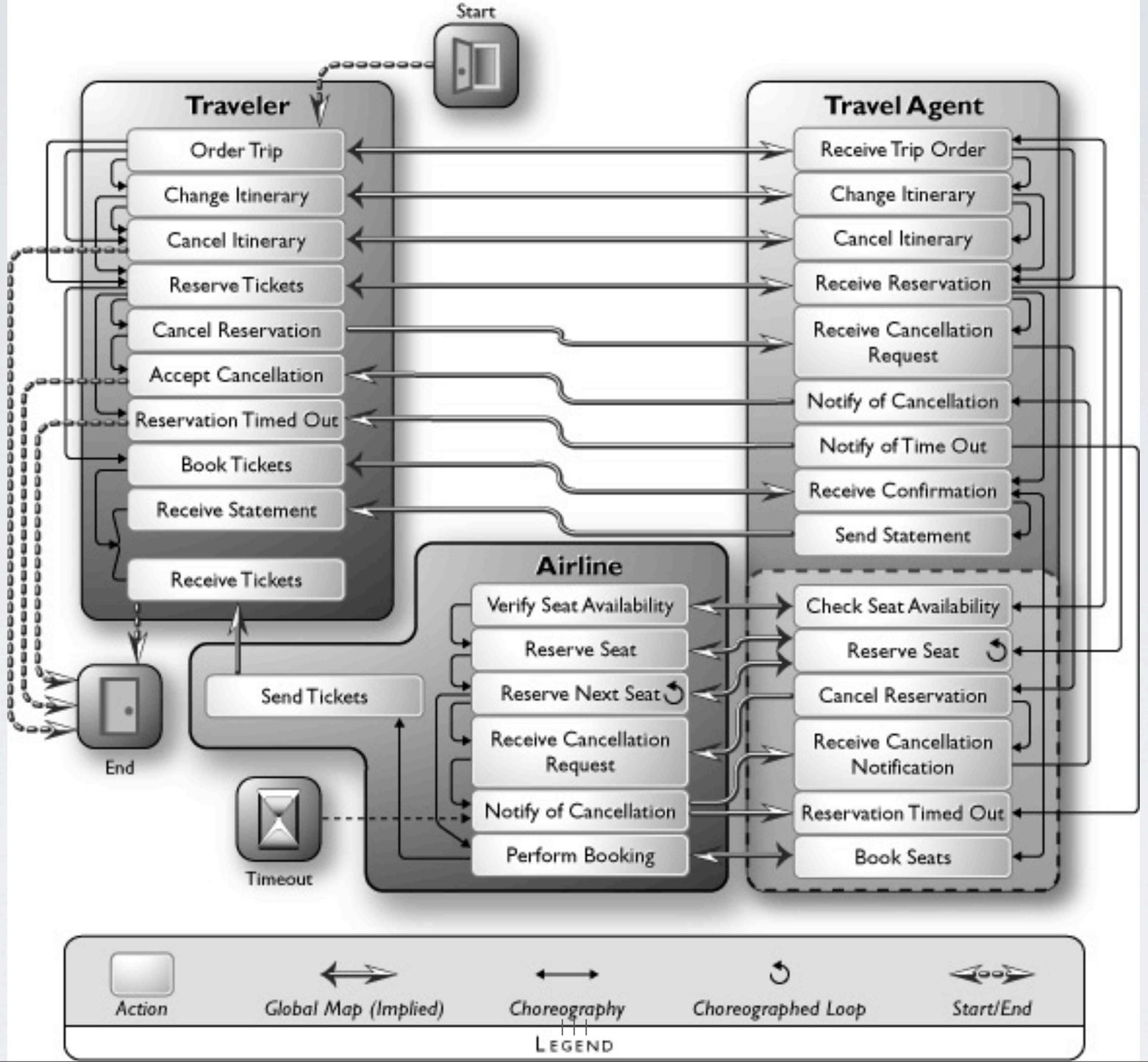
```
<interface name="Airline">
 (...)
 <choice>
      <onMessage>
        <action name = "PerformBooking"
                role ="tns:Airline"
                operation = "tns:AirlineToTA/BookSeats">
          <correlate correlation="defs:reservationCorrelation"/>
        </action>

         <action name="SendTickets"
                role ="tns:Airline"
                operation="tns:AirlineToTraveler/SendTickets"/>
      </onMessage>

      <onMessage>
        <action name = "ReceiveCancellationRequest">
                role ="tns:Airline"
                operation="tns:AirlineToTA/CancelReservation">
          <correlate correlation="defs:reservationCorrelation"/>
        </action>
        <compensate name = "CompensateReservation"
                    transaction = "tns:seatReservation"/>
      </onMessage>
 </choice>
```

O MODELO GLOBAL DE WSCI

- O modelo global é representado
  - pelo conjunto de interfaces dos vários serviços participantes da coreografia e
  - por uma coleção de ligações conectando as operações dos serviços; ligações indicam que os serviços irão trocar mensagens entre si.
- O modelo global permite
  - visualização, análise,
  - verificação e validação,
  - simulação

MODELO GLOBAL

```
<wsdl:definitions   name = "Traveler"
   targetNamespace = "http://example.com/consumer/models"
        xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
        xmlns = "http://www.w3.org/2002/07/wsci10"
        xmlns:tra = "http://example.com/consumer/traveler"
        xmlns:air = "http://example.com/consumer/airline"
        xmlns:ta = "http://example.com/consumer/travelagent">

   <wsdl:import   namespace = "http://example.com/consumer/traveler"
          location = "http://example.com/traveler.wsci" />
   <wsdl:import   namespace = "http://example.com/consumer/airline"
          location = "http://example.com/airline.wsci" />
   <wsdl:import   namespace = "http://example.com/consumer/travelagent"
          location = "http://example.com/travelagent.wsci" />

   <model name = "AirlineTicketing">

     <interface ref = "air:Airline" />
     <interface ref = "tra:Traveler" />
     <interface ref = "ta:TravelAgent" />
     <!-- Traveler / TravelAgent -->
     <connect operations =    "tra:TravelerToTA/PlaceItinerary
                              ta:TAtoTraveler/ReceiveTrip" />
     <!-- other connect elements -->
   </model>
</wsdl:definitions>
```


Start
Traveler
Order Trip
Change Itinerary
Cancel Itinerary
Reserve Tickets
Cancel Reservation
Accept Cancellation
Reservation Timed Out
Book Tickets
Receive Statement
Receive Tickets
Travel Agent
Receive Trip Order
Change Itinerary
Cancel Itinerary
Receive Reservation
Receive Cancellation Request
Notify of Cancellation
Notify of Time Out
Receive Confirmation
Send Statement
Airline
Verify Seat Availability
Reserve Seat
Reserve Next Seat
Receive Cancellation Request
Notify of Cancellation
Perform Booking
Send Tickets
Check Seat Availability
Reserve Seat
Cancel Reservation
Receive Cancellation Notification
Reservation Timed Out
Book Seats
End
Timeout
Action
Global Map (Implied)
Choreography
Choreographed Loop
Start/End
LEGEND

WS-CDL

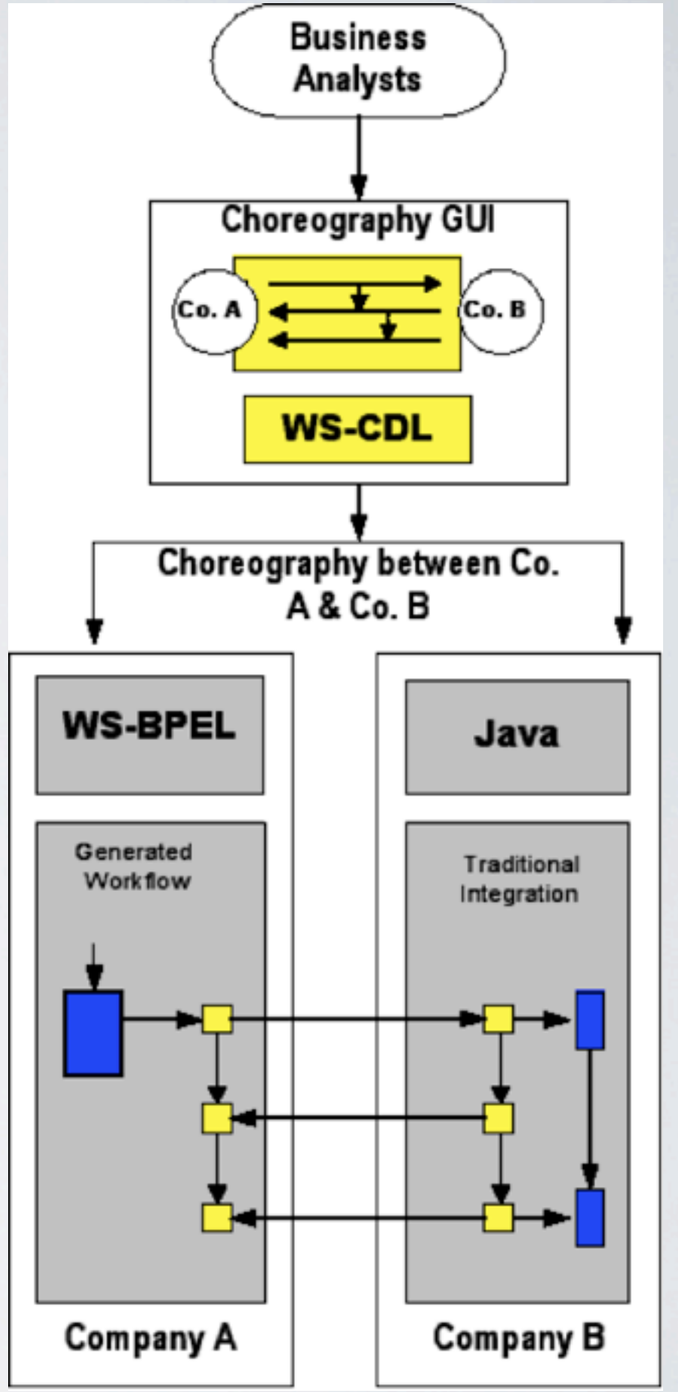
- Desenvolvido pelo W3C WS *Choreography working group* de 2003 a 2006.
- Proposta de padrão encaminhada 11/2005.
- Elaborada por Oracle, Novell, Adobe, W3C, etc.
- É a proposta de padrão mais recente.
- Porém ainda pouco estudada e implementada.
- Pode ser o caminho para o futuro (?)
- http://www.w3.org/TR/ws-cdl-10

WS-CDL

- Permite a definição de colaborações de longa duração entre serviços em ambientes heterogêneos
  - implementados em diferentes linguagens, tecnologias e modelos de programação
  - fazendo parte de várias organizações e domínios.
- Define comportamento comum e particular a partir de um ponto de vista global.
- A Coreografia é um contrato de comum acordo entre os vários participantes da colaboração.

WS-CDL

- Colaboração é definida de um ponto de vista global.
- Diferentes organizações concordam na coreografia que as interligará mas não precisam expor seus processos internos.
- Não é uma linguagem de processos de negócio (BPEL), mas um participante pode usar uma BPEL para implementar seus processos.



WS-CDL

```
<package
  name="NCName"  author="xsd:string"? version="xsd:string"?
  targetNamespace="uri" xmlns="http://www.w3.org/2005/10/cdl">
  <informationType/>*
  <token/>*
  <tokenLocator/>*
  <roleType/>*
  <relationshipType/>*
  <participantType/>*
  <channelType/>*
  Choreography-Notation*
</package>
```

informação trocada entre participantes,
um token é uma referência para parte de uma variável

tipos de participantes da coreografia

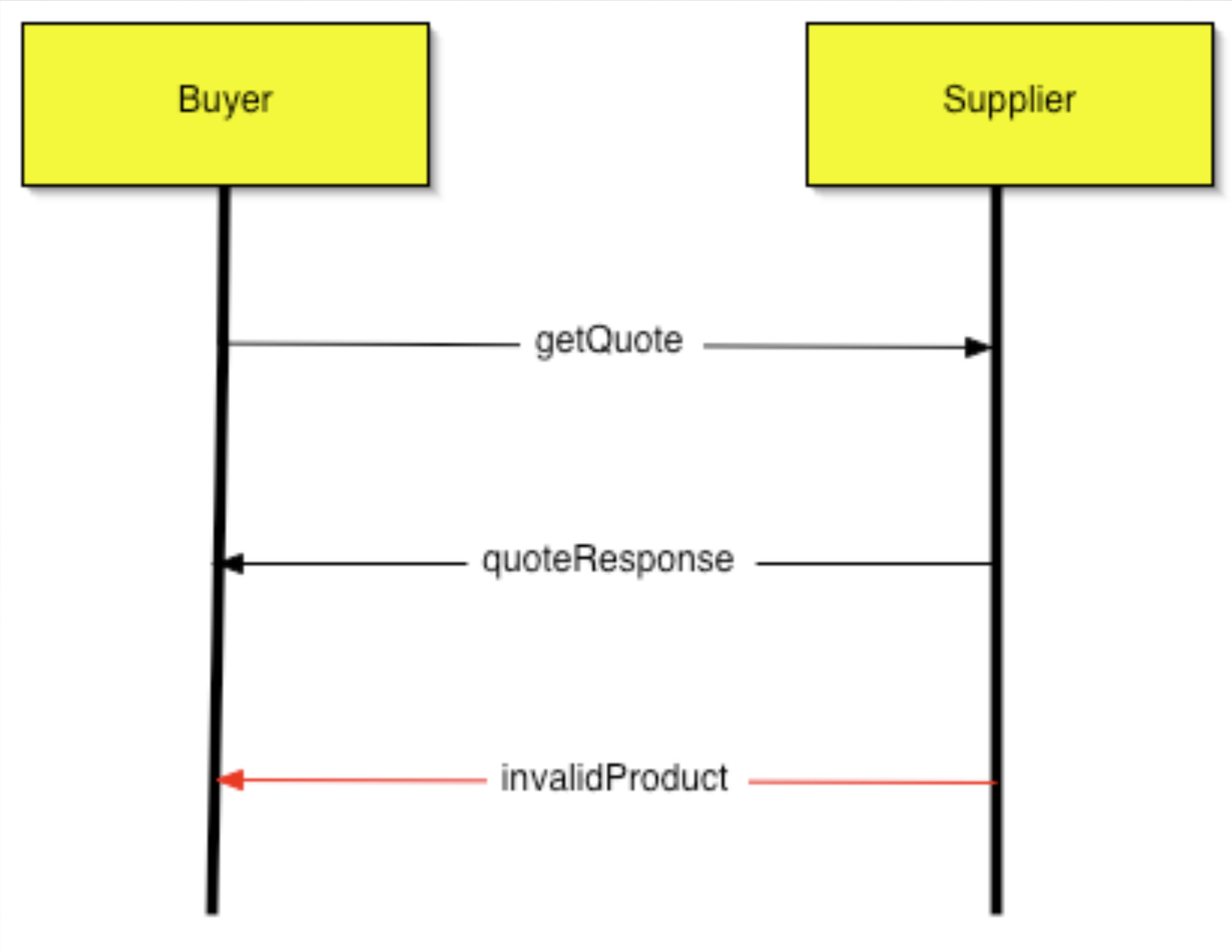
onde e como a informação é trocada

interações entre os participantes

DEFINIÇÃO DA COREOGRAFIA

- São necessários 3 passos:
  - declarar os relacionamentos (que atuam como uma checagem de tipos nos canais de comunicação);
  - declarar variáveis para as instâncias dos canais e dos tipos de informação a serem transmitidas;
  - definir as interações e restrições de ordenação entre as interações.
- vamos ver um pequeno exemplo simplificado

EXEMPLO SIMPLIFICADO



DEFINIÇÃO DAS INTERAÇÕES E VARIÁVEIS

```
<interaction name="QuoteElicitation" operation="getQuote" channelVariable="tns:Buyer2SellerC">
    <description type="documentation">
        Quote Elicitation
    </description>
    <participate relationshipType="tns:Buyer2Seller" fromRoleTypeRef="tns:BuyerRole"
                 toRoleTypeRef="tns:SellerRole"/>
    <exchange name="QuoteRequest" informationType="tns:QuoteRequestType" action="request">
        <description type="documentation">Quote Request Message Exchange</description>
        <send variable="cdl:getVariable('quoteRequest','','')"/>
        <receive variable="cdl:getVariable('quoteRequest','','')"/>
    </exchange>
    <exchange name="QuoteResponse" informationType="tns:QuoteResponseType" action="respond">
        <description type="documentation">Quote Response Message Exchange</description>
        <send variable="cdl:getVariable('quoteResponse','','')"/>
        <receive variable="cdl:getVariable('quoteResponse','','')"/>
    </exchange>
    <exchange name="QuoteResponseFault" informationType="tns:QuoteResponseFaultType"
              action="respond" faultName="InvalidProductFault">
        <description type="documentation">Quote Response Fault Exchange</description>
        <send variable="cdl:getVariable('faultResponse','','')"/>
        <receive variable="cdl:getVariable('faultResponse','','')"/>
    </exchange>
</interaction>
```

```
<choreography name="DegenerateChoreography" root="true">
    <description type="documentation">
        The Choreography for the degenerate use case
    </description>
    <relationship type="tns:Buyer2Seller"/>
    <variableDefinitions>
        <variable name="Buyer2SellerC"
            channelType="tns:Buyer2SellerChannel"
            roleTypes="tns:BuyerRole tns:SellerRole">
            <description type="documentation"> Channel Variable</description>
        </variable>
        <variable name="quoteRequest"
            informationType="tns:QuoteRequestType"
            roleTypes="tns:BuyerRole tns:SellerRole">
            <description type="documentation">Request Message</description>
        </variable>
        <variable name="quoteResponse"
            informationType="tns:QuoteResponseType"
            roleTypes="tns:BuyerRole tns:SellerRole">
            <description type="documentation">Response Message</description>
        </variable>
        <variable name="faultResponse"
            informationType="tns:QuoteResponseFaultType"
            roleTypes="tns:BuyerRole tns:SellerRole">
            <description type="documentation">Fault Message</description>
        </variable>
    </variableDefinitions>
```

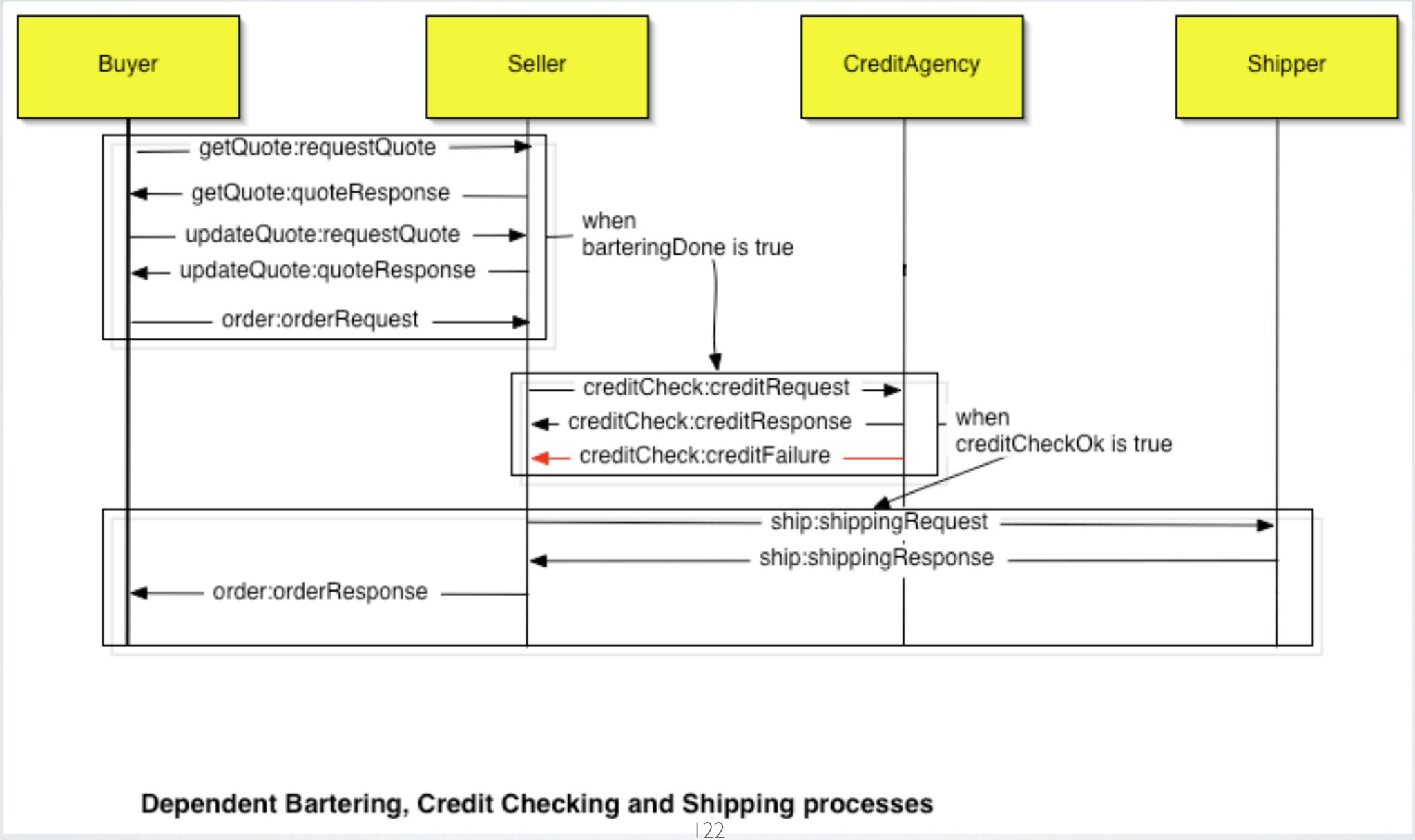
```
<choreography name="DegenerateChoreography" root="true">
        <description type="documentation">
            The Choreography for the degenerate use case
        </description>
        …
      <sequence>
        <interaction name="QuoteElicitation" operation="getQuote" channelVariable="tns:Buyer2SellerC">
            <description type="documentation">
                Quote Elicitation
            </description>
            <participate relationshipType="tns:Buyer2Seller" fromRoleTypeRef="tns:BuyerRole"
                         toRoleTypeRef="tns:SellerRole"/>
            <exchange name="QuoteRequest" informationType="tns:QuoteRequestType" action="request">
                <description type="documentation">Quote Request Message Exchange</description>
                <send variable="cdl:getVariable('quoteRequest','','')"/>
                <receive variable="cdl:getVariable('quoteRequest','','')"/>
            </exchange>
            <exchange name="QuoteResponse" informationType="tns:QuoteResponseType" action="respond">
                <description type="documentation">Quote Response Message Exchange</description>
                <send variable="cdl:getVariable('quoteResponse','','')"/>
                <receive variable="cdl:getVariable('quoteResponse','','')"/>
            </exchange>
            <exchange name="QuoteResponseFault" informationType="tns:QuoteResponseFaultType"
                      action="respond" faultName="InvalidProductFault">
                <description type="documentation">Quote Response Fault Exchange</description>
                <send variable="cdl:getVariable('faultResponse','','')"/>
                <receive variable="cdl:getVariable('faultResponse','','')"/>
            </exchange>
        </interaction>
      </sequence>
    </choreography>
</package>
```

COREOGRAFIAS MAIS COMPLEXAS

- Uma coreografia tem um nome e uma raiz (ponto de entrada) e pode ter estruturas compostas por
  - sequências
  - paralelização
  - escolhas
  - unidades de trabalho
  - interações
  - atribuição a variáveis
  - ações silenciosas e noAction
  - ou a execução de uma sub-coreografia
- Ainda pode ter um Bloco de Exceção e um Bloco de Finalização

COREOGRAFIAS MAIS INTERESSANTES

SERÃO MAIS COMPLEXAS...



Dependent Bartering, Credit Checking and Shipping processes

DESAFIOS DE PESQUISA (1/5)

- Projeto Baile - HP
- Projeto CHOReOS - European Commission FP7
- Realização de Coreografias de Grande Escala
  - definição
  - execução
  - análise
  - monitoramento
  - depuração

DESAFIOS DE PESQUISA (2/5)

- Verificação e Validação
  - testes
  - desenvolvimento dirigido por testes
  - monitoração em tempo de execução
- Modelagem analítica
  - modelos matemáticos para
    - avaliar propriedades / corretude
    - prever desempenho/escalabilidade
- Simulação

DESAFIOS DE PESQUISA (3/5)

- *Middleware* para realização (*enactment*)
- Metodologia de desenvolvimento
- Ferramentas
- Mapeamento para
  - Computação em Grades
  - Computação em Nuvem

DESAFIOS DE PESQUISA (4/5)

- Adaptação dinâmica
- Gerenciamento de dependências
- Gerência de Qualidade de Serviço
- Internet das coisas / Computação ubíqua

DESAFIOS DE PESQUISA (5/5)

- Aplicações-exemplo concretas
  - coreografias sintéticas
    - para testes e avaliação de desempenho e escalabilidade
  - Aeroporto amigável
  - Rede de taxis inteligentes
  - Coordenação móvel de pessoas
    - jornalismo dos cidadãos
    - coordenação de um brainstorming de grande escala

PERGUNTAS E DISCUSSÃO

- Slides disponíveis em:
  - www.ime.usp.br/~kon/presentations.html
  - www.ccsl.ime.usp.br/baile

REFERÊNCIAS

- W3C Web Services Choreography Working Group Charter
- http://www.w3.org/2005/12/wscwg-charter.html
- Web Services Glossary, http://www.w3.org/TR/ws-gloss/
- Bruno Martins Duarte dos Santos. Web Services Choreography. 2007. Disponível em http://www.webartigos.com/articles/2901/1/Web-Services-Choreography
- W3C Web Services Choreography Description Language: Primer, 2006. http://www.w3.org/TR/2006/WD-ws-cdl-10-primer-20060619/
- W3C Web Service Choreography Interface (WSCI) 1.0. 2002. http://www.w3.org/TR/wsci
- TILKOV, Stefan. A Brief Introduction to REST, http://www.infoq.com/articles/rest-introduction , 2007

REFERÊNCIAS

- Burke, Putting Java to REST, http://java.dzone.com/articles/putting-java-rest, 2008
- ZHANG, Liang-Jie. Services Computing, Pequim, Springer, 2007
- Hewitt, Eben, Java SOA Cookbook, 1st Edition, 2009,
- Shin, San, SOA Using OpenESB, BPEL and NetBeans, 2010
- http://ccsl.ime.usp.br/baile
- http://ccsl.ime.usp.br/wiki/index.php/Baile