COMP3017
Service Computing

Where to work on your hot topic study - 1

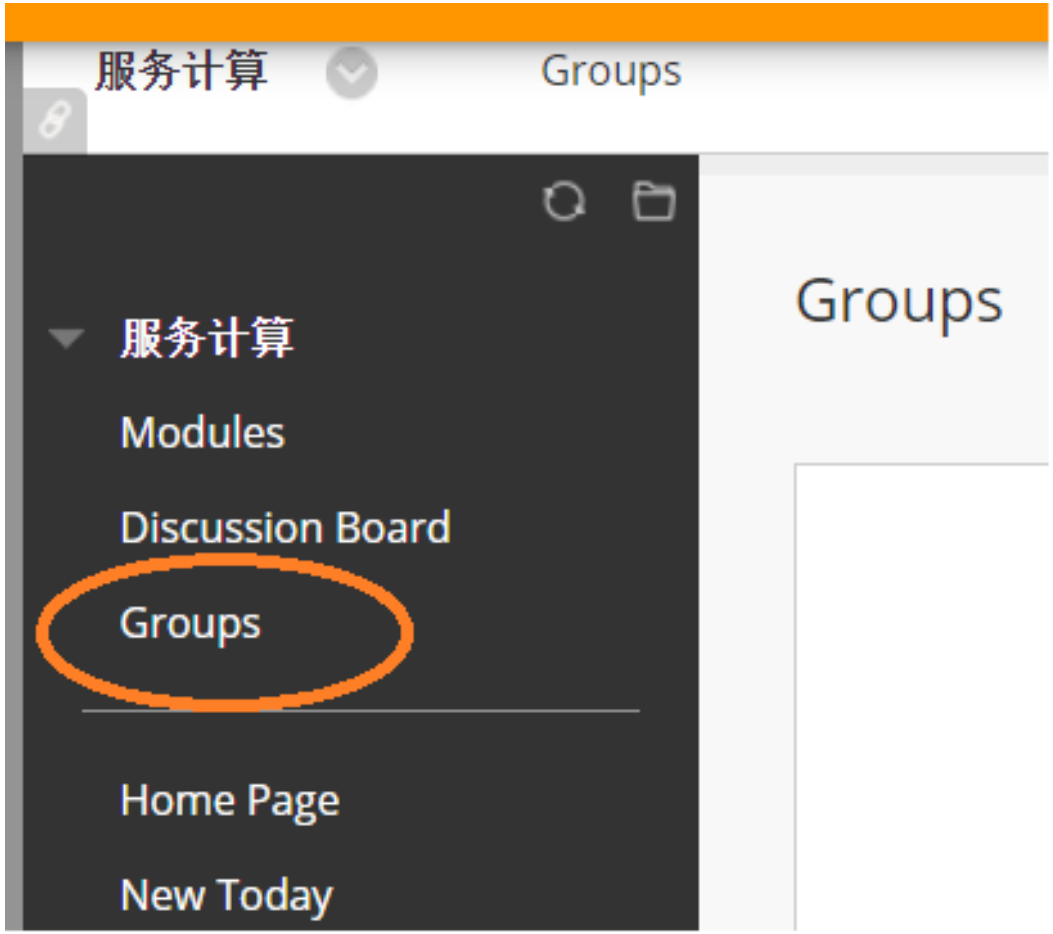
- Internet
  - Find good resources about your topic
    - Web pages
    - Articles
    - Research papers
    - Videos

Hot topic study – task 1 – individual work

- Who – individually
- How - Consider your group's topic, find several online resources about it (web pages, articles, videos, figures, etc.), write a short overview/your understanding of the group's topic
- What to submit
  - The resources you found (at least 2 in English, optional in Chinese)
  - Short overview of the topic written by you
- Where to submit:
  - Blackboard/ your group's area / "Journal"
- When to submit – by Sunday, 27th March

Where to work on your hot topic study - 2

- Blackboard **Groups** area
  - To submit all your milestones

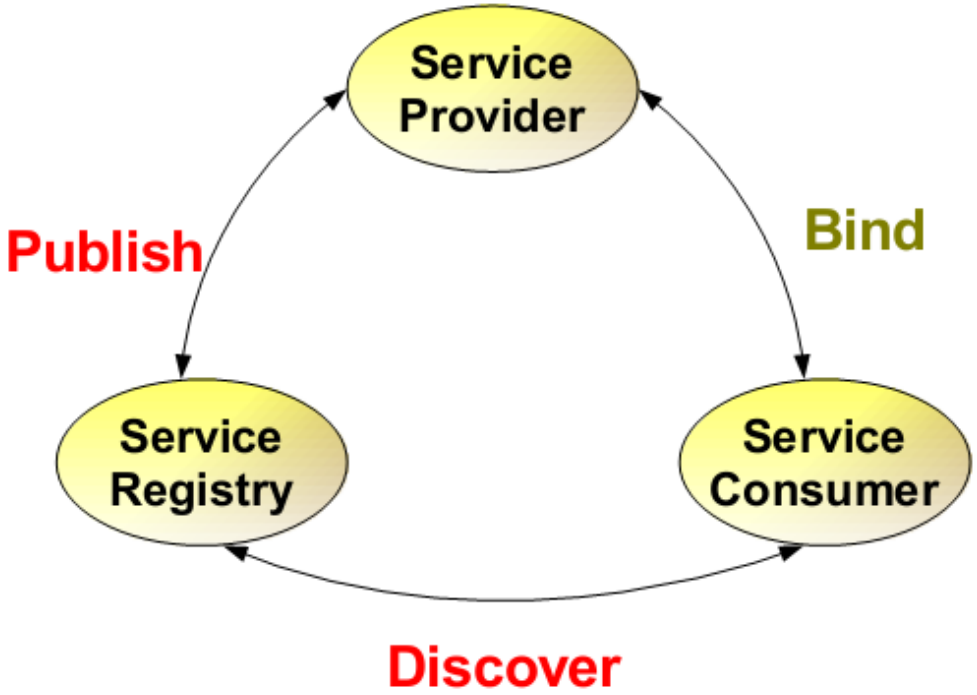


Module Three: Service Discovery UDDI and Service Composition BPEL

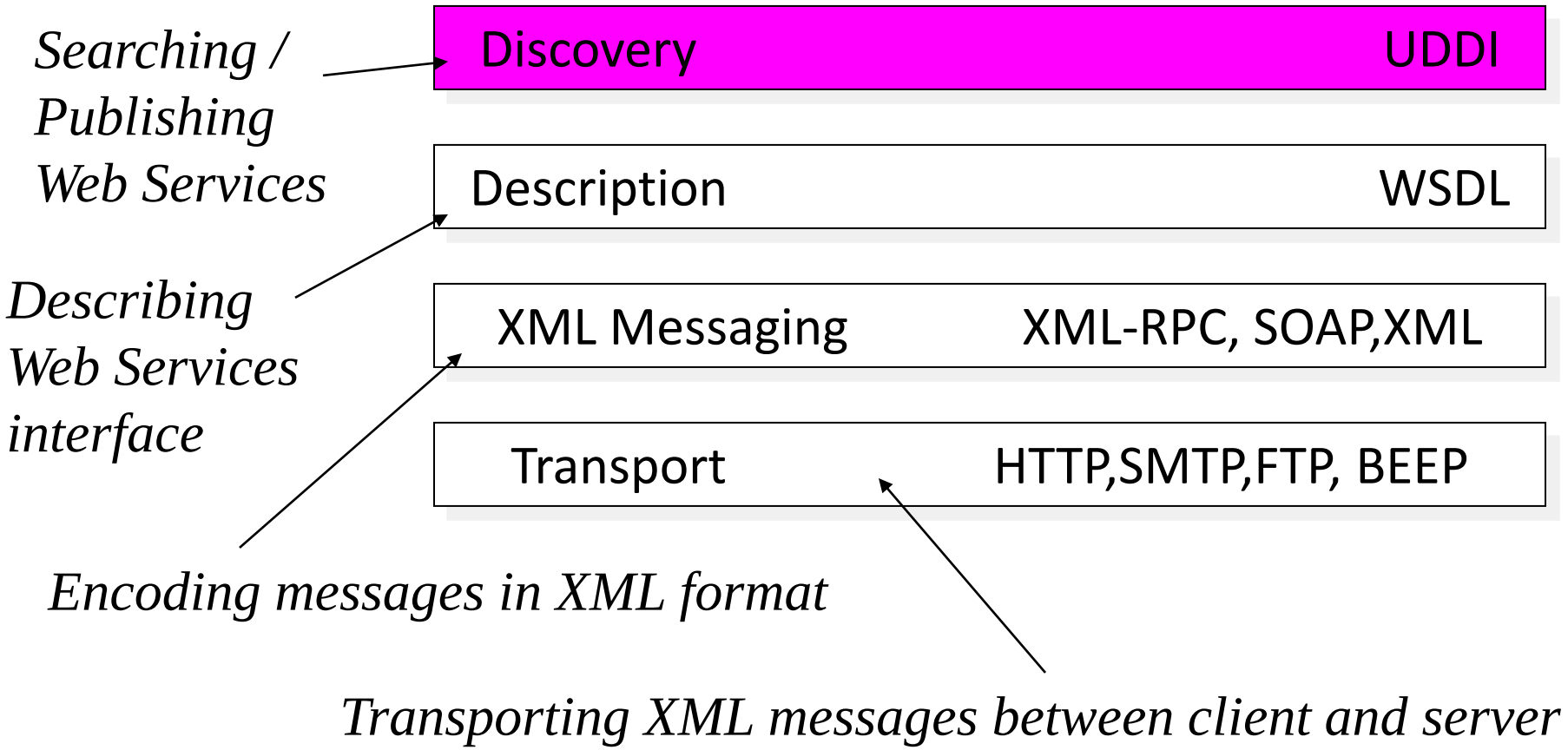
Service Discovery UDDI

Service Architecture

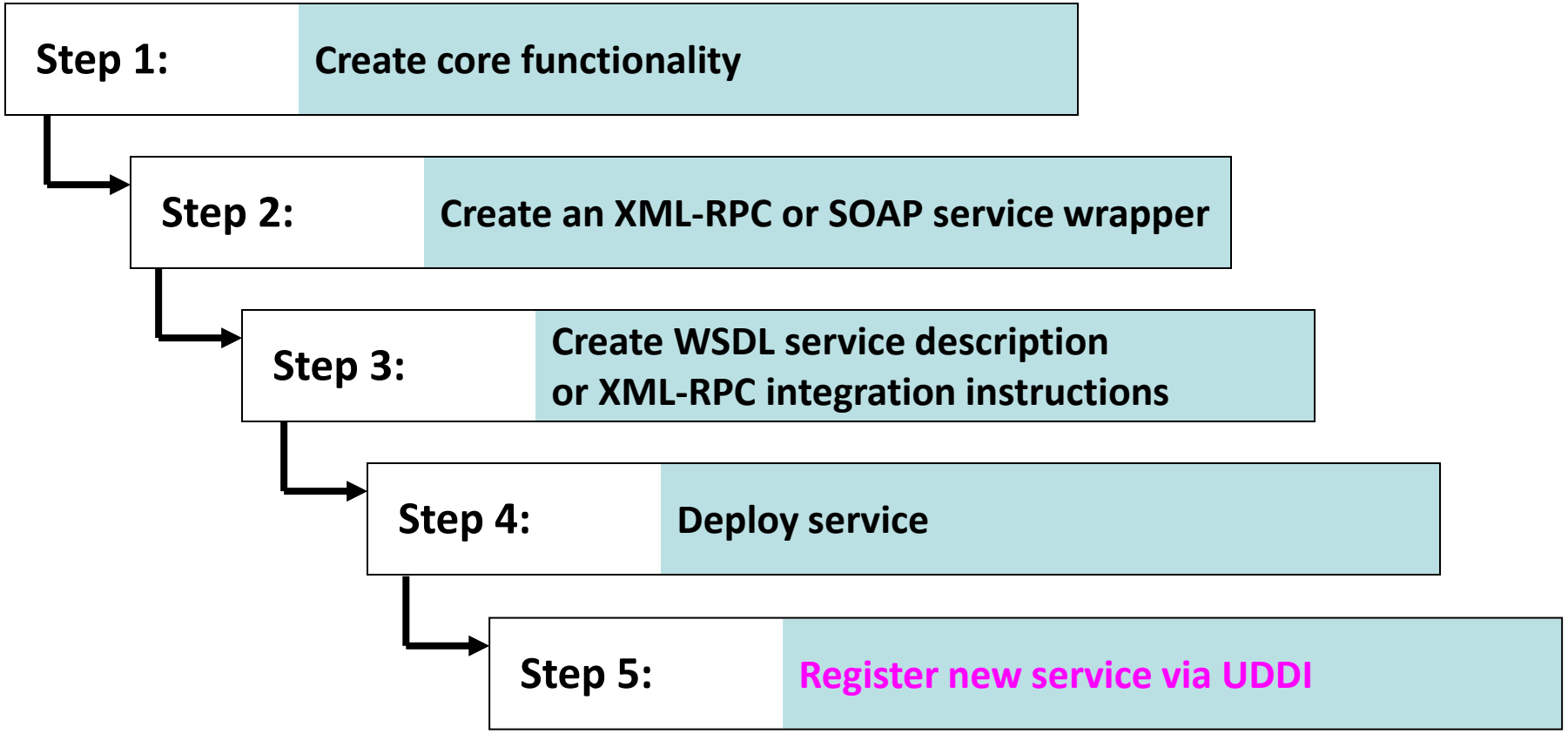
- UDDI(Universal Description, Discovery, and Integration) defines a scheme to publish and discover information about Web services



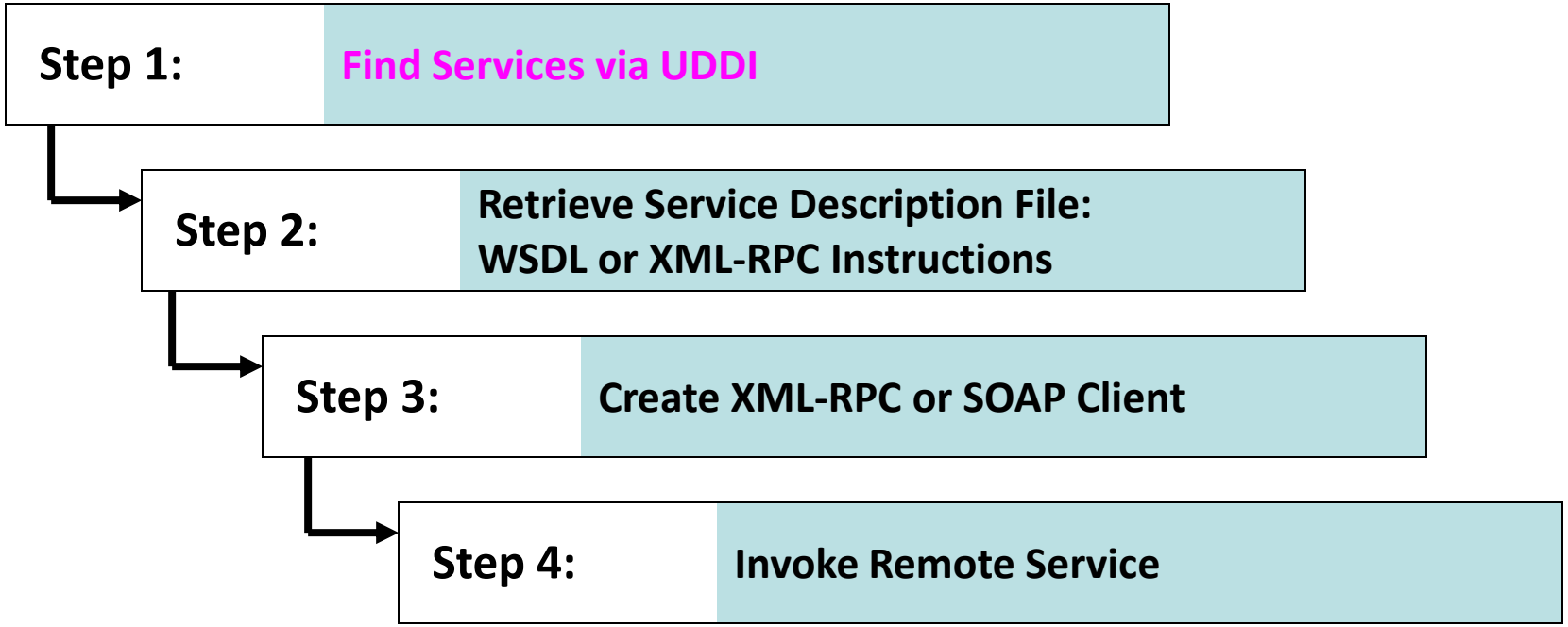
Web Service Protocol Stack



Using the Protocols Together – service provider perspective



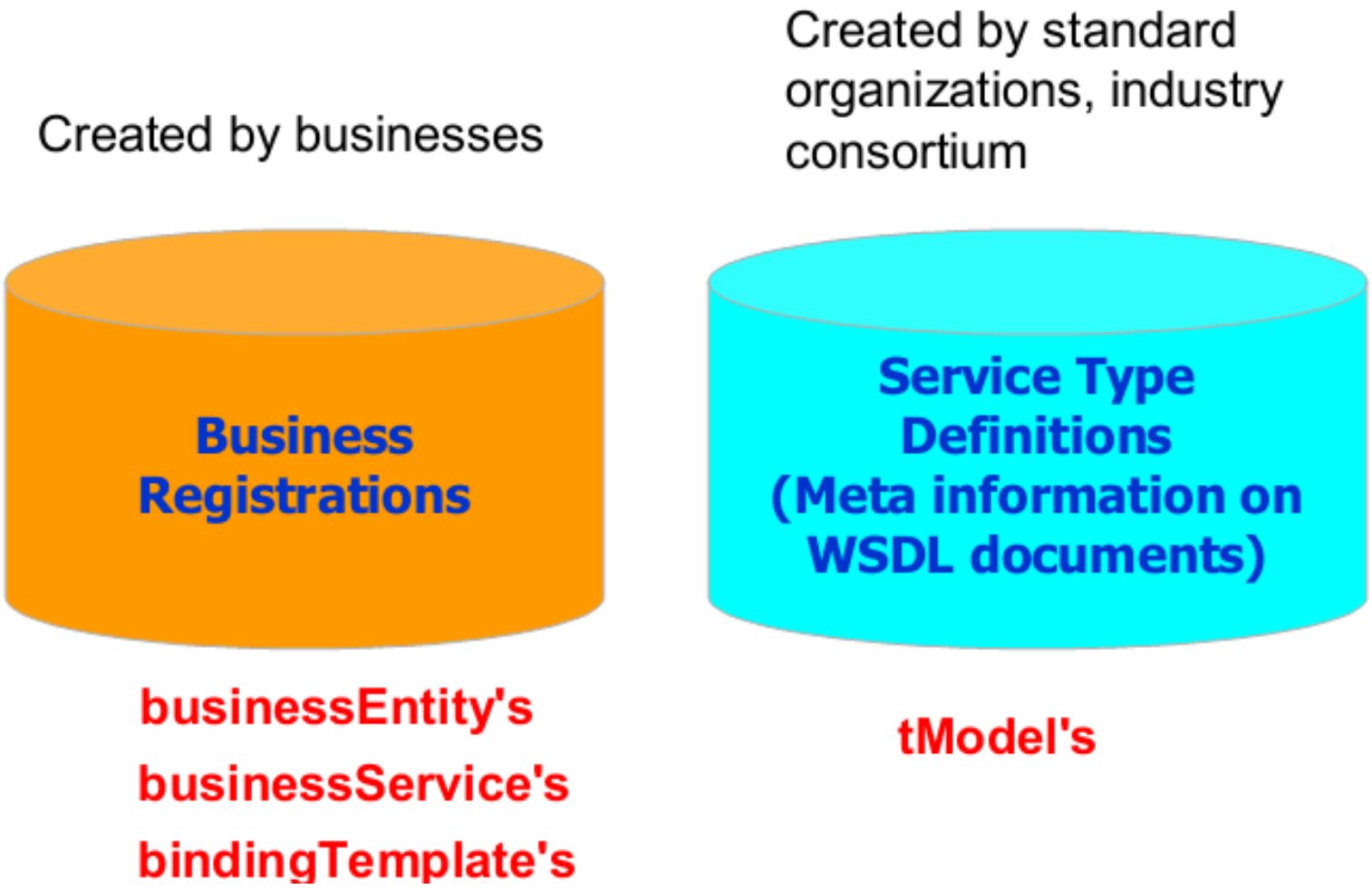
Using the Protocols Together – service request perspective



UDDI Adoption Phases

- Phase 1: Experimental stage
- Phase 2: Private UDDI registry within an intranet
- Phase 3: Public UDDI registries with no coordination among them
- Phase 4: Public UDDI registries with coordination (i.e. replication)
- Phase 5: Value added registry services

Registry Data



Issues of UDDI

- How do you know if the data you get is valid, legitimate, and up to date?
- How do you measure quality of data?
- How do you make sure only the qualified entities register their service information (authentication)?
- How do you provide access control to the data in the registry?
- How do you synchronize the data in multi-registry environment?

Open Question Points: 10

Setting

In a 2000 vision, the **publicly operated** UDDI node or broker would return services listed for public discovery by others. We now know that this vision has failed. Do you know about alternative way to implement and use UDDI registries?

Answer

UDDI implementations

- Public federated network of UDDI registries available on the Internet

UDDI implementations

- Public federated network of UDDI registries available on the Internet
  -
- Private UDDI registries developed by companies or industry groups

UDDI implementations

- Public federated network of UDDI registries available on the Internet
  - UDDI registry is a part of the global federated network
- Private UDDI registries developed by companies or industry groups
  - UDDI registry is privately owned and operated
  - Allows members of the company or the industry group to share and advertise services amongst themselves

UDDI implementations

- Public federated network of UDDI registries available on the Internet
  - UDDI registry is a part of the global federated network
- Private UDDI registries developed by companies or industry groups
  - UDDI registry is privately owned and operated
  - Allows members of the company or the industry group to share and advertise services amongst themselves

- In common: Web services API for publishing and locating businesses and services advertised within the UDDI registry

UDDI current use

- Support SOA governance within an organization
- Offering a standardized way to
  - catalog company matadata
  - categorize it in multiple dimensions
- Used to dynamically bind client systems to implementations

Example – UDDI registry in a large enterprise

- A “Widget, Ltd” company has an existing application called “OurEmployeesHR” that provides telephone numbers and human resources (HR) information about employees

Example – UDDI registry in a large enterprise

- A “Widget, Ltd” company has an existing application called “OurEmployeesHR” that provides telephone numbers and human resources (HR) information about employees
- You are a developer in the same company. You want to write an application called “Procurement” for a procurement function that also needs to provide HR information to the supplier
  - Your application needs to give the supplier access to the employee account codes after the employee provides a name or serial number

Open Question Points: 10

Setting

Consider scenario before the Web services. Can you reuse the existing "OurEmployeesHR" application while writing your own "Procurement" application? What problems do you see? (you cannot use Web services to help you)

Answer

Example (cont)

- Before the web services
  - The developer does not know about the similar application

Example (cont)

- Before the web services
  - The developer does not know about the similar application
  - The developer knows about the application but cannot reuse it because of technical barriers

Example (cont)

- Before the web services
  - The developer does not know about the similar application
  - The developer knows about the application but cannot reuse it because of technical barriers
  - The developer knows about the application and reuses it, but only after significant time and negotiation

Open Question Points: 10

Setting

(You will have 10 minutes to answer this question, please provide as many details as you can.)

Now consider a scenario where the "Widget, Ltd" decides to use Web services. Describe how a developer of "Procurement" application can reuse "OurEmployeesHR" application.

What steps must be taken? How is UDDI used here? What would be the benefits for the developer and for the company?

Answer

Example (cont)

- With UDDI
  - “OurEmployeesHR” application is turned into a web service and published to the registry

Example (cont)

- With UDDI
  - “OurEmployeesHR” application is turned into a web service and published to the registry
  - The developer can search for the web service and reuse the existing technical component in their new application for the supplier in minutes

Example (cont)

- With UDDI
  - “OurEmployeesHR” application is turned into a web service and published to the registry
  - The developer can search for the web service and reuse the existing technical component in their new application for the supplier in minutes
  - The developer saves the time and gets the application running sooner
    - Increased efficiency
    - Saved time and money

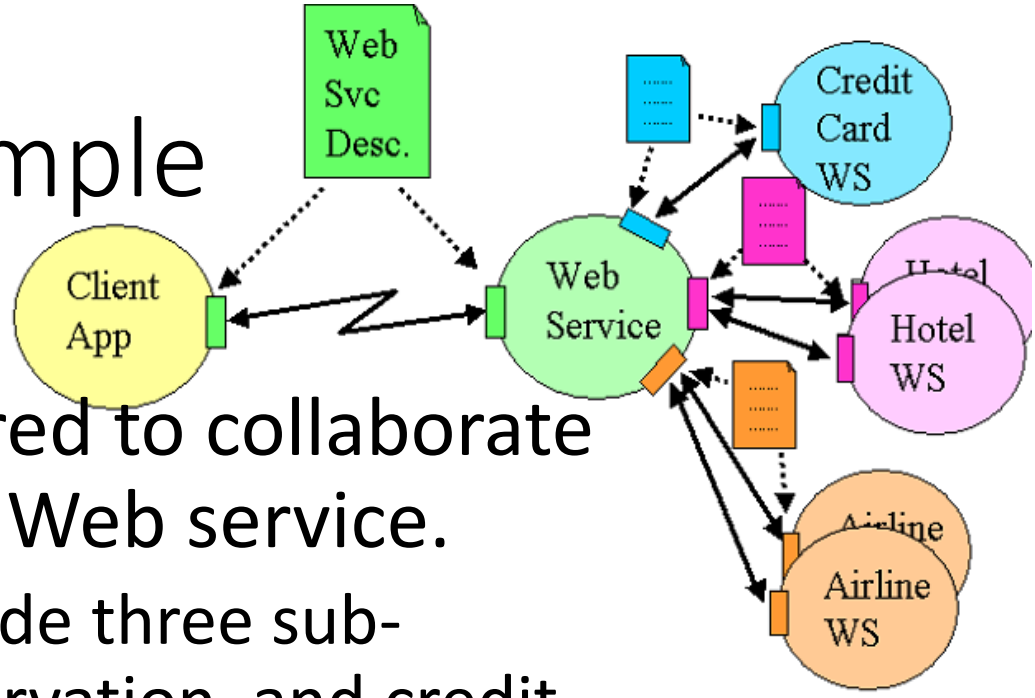
- Even though UDDI isn’t widely used, the concepts behind it are still very relevant

Service Composition BPEL

What is service composition?

- Service composition allows developers to “compose” services that exchange SOAP messages and define their interfaces into an aggregate solution.
- The aggregate is a composed Web service or a so-called composite Web service.

Composite Web Service example



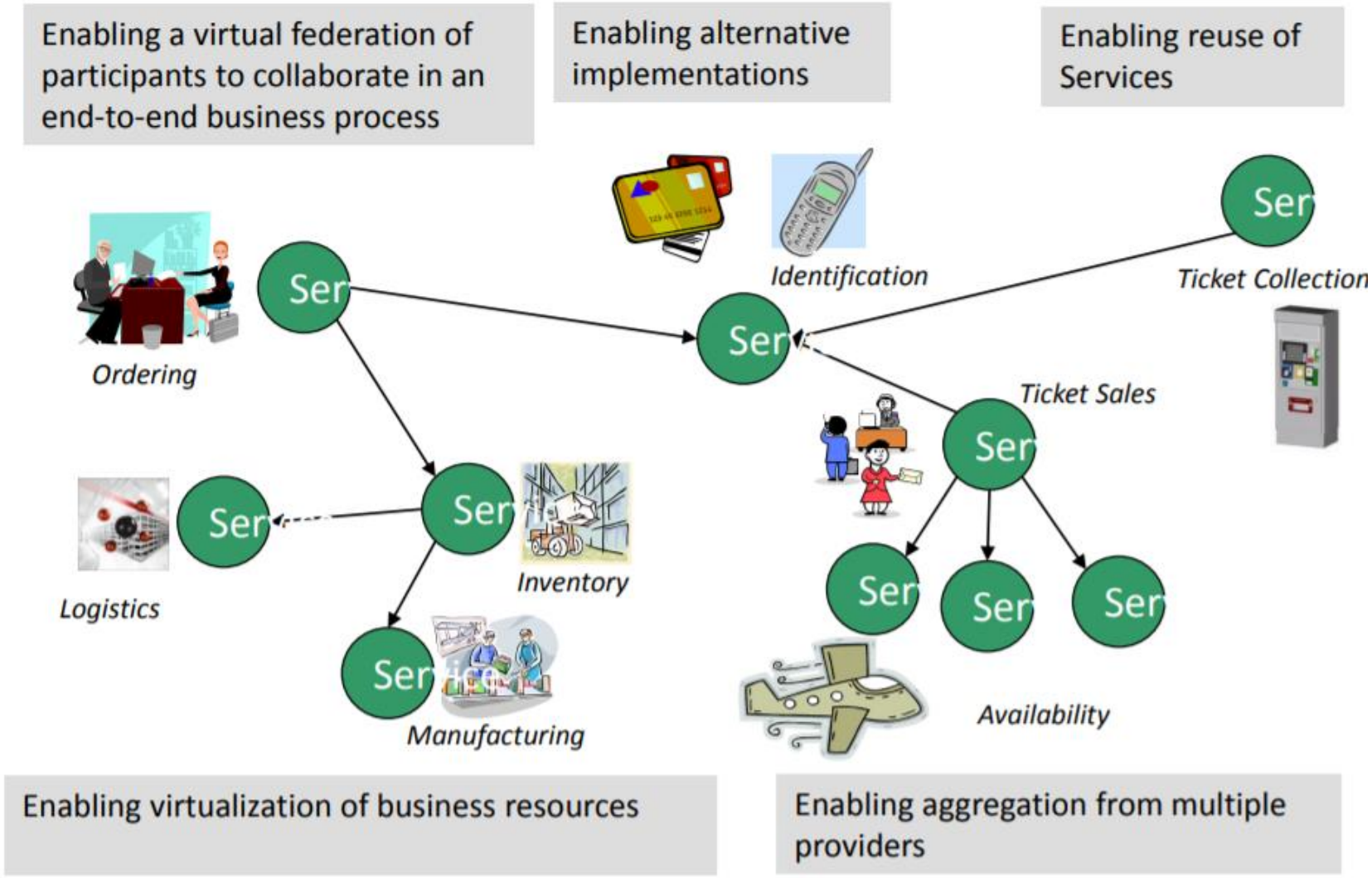
- Multiple Web services may be required to collaborate with each other to form a composite Web service.
  - A travel booking Web service may include three sub-processes: flight reservation, hotel reservation, and credit card payment.
  - These three sub-processes may be performed by three individual Web services provided by corresponding service providers.
  - The travel booking Web service thus becomes a composite Web service involving three collaborative Web services.

Business Process

- Business companies are driven by underlying business processes
- Business process is a set of activities that are coordinated to achieve a certain business goal.
- Business process is a structured and measurable set of activities that consume certain resources and are designed to produce the specified output for a particular business requirement.

Motivation - ENABLE FLEXIBLE, FEDERATED BUSINESS PROCESSES

- Enable flexible, federated business processes

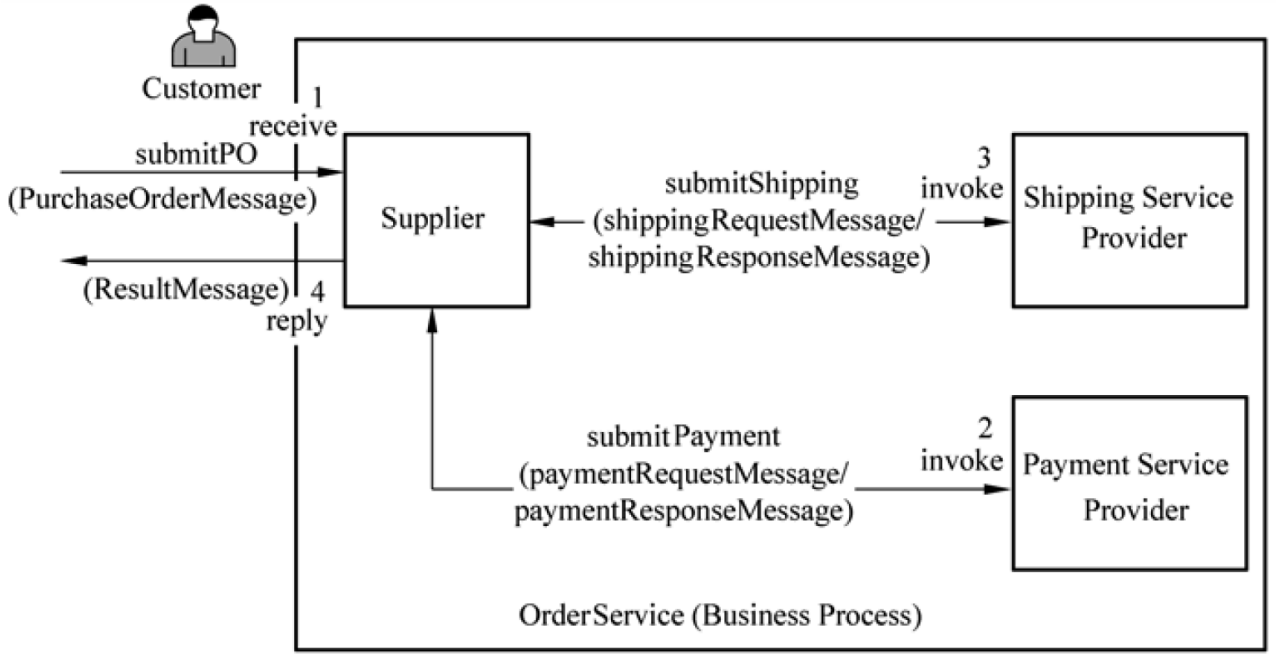


What is BPEL?

- The Business Process Execution Language for Web Services is such a flow representation developed to facilitate coordination of Web services into a comprehensive business process.
- In short: BPEL is a business process language that can be used to represent composite services.

OrderService business process example

- The customer calls the *submitPO* operation with a message *PurchaseOrderMessage*;
- The supplier calls the *submitPayment* operation with a message *paymentRequestMessage* and receives a message *paymentResponseMessage*
- The supplier calls the *submitShipping* operation with a message *shippingRequestMessage* and receives a message *shippingResponseMessage*;
- The supplier returns to the customer a message *ResultMessage*.



BPEL definition sections

- Partner link definition
- Variables
- Process definition

An example BPEL definition for a business order process

```
<process name="orderService"
   targetNamespace="http://servicescomputing.org/bpel4ws/purchase"
   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
   xmlns:lns="http://servicescomputing.org/wsdl/po">

<!-- Define partner links -->
   <partnerLinks>
      <partnerLink name="purchasing"
                   partnerLinkType="lns:supplyLT"
                   myRole="purchaseService"/>
      <partnerLink name="payment"
                   partnerLinkType="lns:paymentLT"
                   myRole="paymentRequestor"
                   partnerRole="paymentServiceRequestor"/>
      <partnerLink name="shipping"
                   partnerLinkType="lns:shippingLT"
                   myRole="shippingRequestor"
                   partnerRole="shippingServiceRequestor"/>
  </partnerLinks>
```

An example BPEL definition for a business order process

```
<process name="orderService"
   targetNamespace="http://servicescomputing.org/bpel4ws/purchase"
   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
   xmlns:lns="http://servicescomputing.org/wsdl/po">

<!-- Define partner links -->
   <partnerLinks>
      <partnerLink name="purchasing"
                   partnerLinkType="lns:supplyLT"
                   myRole="purchaseService"/>
      <partnerLink name="payment"
                   partnerLinkType="lns:paymentLT"
                   myRole="paymentRequestor"
                   partnerRole="paymentServiceRequestor"/>
      <partnerLink name="shipping"
                   partnerLinkType="lns:shippingLT"
                   myRole="shippingRequestor"
                   partnerRole="shippingServiceRequestor"/>
  </partnerLinks>
```

An example BPEL definition for a business order process

```
<process name="orderService"
   targetNamespace="http://servicescomputing.org/bpel4ws/purchase"
   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
   xmlns:lns="http://servicescomputing.org/wsdl/po">

<!-- Define partner links -->
   <partnerLinks>
      <partnerLink name="purchasing"
                   partnerLinkType="lns:supplyLT"
                   myRole="purchaseService"/>
      <partnerLink name="payment"
                   partnerLinkType="lns:paymentLT"
                   myRole="paymentRequestor"
                   partnerRole="paymentServiceRequestor"/>
      <partnerLink name="shipping"
                   partnerLinkType="lns:shippingLT"
                   myRole="shippingRequestor"
                   partnerRole="shippingServiceRequestor"/>
  </partnerLinks>
```

Partner link definition section

- Involved business parties are grouped by a tag *partnerLinks*
- Each partner link is characterized by a tag *partnerLink*.
- Three partner links are defined: *purchasing*, *payment*, and *shipping*.
- The myRole/partnerRole attribute of a partner specifies how the partner and the process interact given the *partnerLinkType*.
  - The *myRole* attribute refers to the role in the serviceLinkType that the process will play
  - The *partnerRole* specifies the role that the partner will play
  - In our example: for the partner *payment,* the supplier acts as a *paymentRequestor* and the payment service provider offers the service; for the partner *shipping,* the supplier acts as a *shippingRequestor* and the shipping service provider offers the service.

Cont.

```
<!-- Define variables -->
  <variables>
     <variable name="PurchaseOrder" messageType="lns:PurchaseOrderMessage"/>
     <variable name="Result" messageType="lns:ResultMessage"/>
     <variable name="ShippingRequest" messageType="lns:shippingRequestMessage"/>
     <variable name="ShippingResponse" messageType="lns:shippingResponseMessage"/>
     <variable name="PaymentRequest" messageType="lns:paymentRequestMessage"/>
     <variable name="PaymentResponse" messageType="lns:paymentResponseMessage"/>
  </variables>
```

The section of *variables*

- Defines the data variables used by the business process, based upon their definitions in terms of WSDL message types, XML Schema simple types, or XML Schema elements.
- For example, the variable *PurchaseOrder* refers to the *PurchaseOrderMessage* defined in the WSDL document; *Result* refers to *ResultMessage*; and *ShippingRequest* refers to *ShippingRequestMessage*.
- Variables allow processes to maintain state data and process history based on messages exchanged

Cont.

```
<!-- Define process -->
  <sequence>
     <receive partnerLink="purchasing"
              portType="lns:purchaseOrderPT"
              operation="submitPurchaseOrder"
              variable="PurchaseOrder">
     </receive>

     <invoke partnerLink="payment"
             portType="lns:paymentPT"
             operation="submitPayment"
             inputVariable="PaymentRequest"
             outputVariable="PaymentResponse">
      </invoke>

      <invoke partnerLink="shipping"
              portType="lns:shippingPT"
              operation="submitShipping"
              inputVariable="ShippingRequest"
              outputVariable="ShippingResponse">
      </invoke>

      <reply partnerLink="purchasing"
             portType="lns:purchaseOrderPT"
             operation="submitPurchaseOrder"
             variable="Result"/>
  </sequence>
</process>
```

Process definition section – receive

- The structure of the main processing section is defined by a pair of *sequence* tags, indicating that four activities are performed sequentially: *receive, payment, shipping,* and *reply*.
- The first activity is a *receive* activity, which accepts incoming customer messages.
  - The definition of a *receive* activity includes the partner who sends the message, the port type, and the operation of the process to which the partner is targeting this message.
- Based on this information, once the process receives a message, it searches for an active *receive* activity that has a matching quadruple <partnerLink, portType, operation, variable> and hands it the message.
- In our example, the *receive* activity invokes the *submitPurchaseOrder* operation from the *purchaseOrderPT* portType with the variable *PurchaseOrder* (i.e., *PurchaseOrderMessage*).

Cont.

```
<!-- Define process -->
  <sequence>
     <receive partnerLink="purchasing"
              portType="lns:purchaseOrderPT"
              operation="submitPurchaseOrder"
              variable="PurchaseOrder">
     </receive>

     <invoke partnerLink="payment"
             portType="lns:paymentPT"
             operation="submitPayment"
             inputVariable="PaymentRequest"
             outputVariable="PaymentResponse">
      </invoke>

      <invoke partnerLink="shipping"
              portType="lns:shippingPT"
              operation="submitShipping"
              inputVariable="ShippingRequest"
              outputVariable="ShippingResponse">
      </invoke>

      <reply partnerLink="purchasing"
             portType="lns:purchaseOrderPT"
             operation="submitPurchaseOrder"
             variable="Result"/>
  </sequence>
</process>
```

Process definition section - invoke

- After the *receive* activity, the process invokes two Web services sequentially, each being delimited using an *invoke* tag.
- First, the process invokes the operation *submitPayment* from the portType *paymentPT*, with an input message *PaymentRequest* (i.e., *PaymentRequestMessage*) and an output message *PaymentResponse* (i.e., *PaymentResponseMessage*).
- Then the process invokes the operation *submitShipping* from the portType *shippingPT*, with an input message *ShippingRequest* (i.e., *ShippingRequestMessage*) and an output message *ShippingResponse* (i.e., *ShippingResponse*Message).

Cont.

```
<!-- Define process -->
  <sequence>
     <receive partnerLink="purchasing"
              portType="lns:purchaseOrderPT"
              operation="submitPurchaseOrder"
              variable="PurchaseOrder">
     </receive>

     <invoke partnerLink="payment"
             portType="lns:paymentPT"
             operation="submitPayment"
             inputVariable="PaymentRequest"
             outputVariable="PaymentResponse">
      </invoke>

      <invoke partnerLink="shipping"
              portType="lns:shippingPT"
              operation="submitShipping"
              inputVariable="ShippingRequest"
              outputVariable="ShippingResponse">
      </invoke>

      <reply partnerLink="purchasing"
             portType="lns:purchaseOrderPT"
             operation="submitPurchaseOrder"
             variable="Result"/>
  </sequence>
</process>
```

Process definition section - reply

- The fourth and the last activity is a *reply* activity, which allows the business process to send a message in reply to the customer.
- Once a reply activity is reached, the quadruple <partnerLink, portType, operation, variable> is used to send the result back to the customer.
- In our example, the *reply* activity invokes the *getResult* operation from the *purchaseOrderPT* portType with the variable *Result* (i.e., *ResultMessage*).
- The combination of a pair of *receive* and *reply* forms a request-response operation on the WSDL portType for the process
  - In this example *submitPurchaseOrder* operation in the portType *purchaseOrderPT*.

Module 3 Summary

- Service discovery
- Service composition

Module 3 Summary

- Service discovery
- Service composition