Module Three: Service Discovery UDDI and Service Composition BPEL

XML-RPC

SOAP

WSDL

UDDI

BPEL

JEE

SOA

Textbooks

- Web Services Essentials
  - Chapter 7
- Services Computing
  - Chapters 3.3-3.6

You can find an online version of this book **for free** through our library webpage

Module 3 Learning Outcomes

- Understand the main concepts of UDDI
- Understand main uses of UDDI,
- Understand the technical aspects of UDDI
- Understand basic concepts of BPEL
- Understand BPEL basic structure
- Be able to create business process

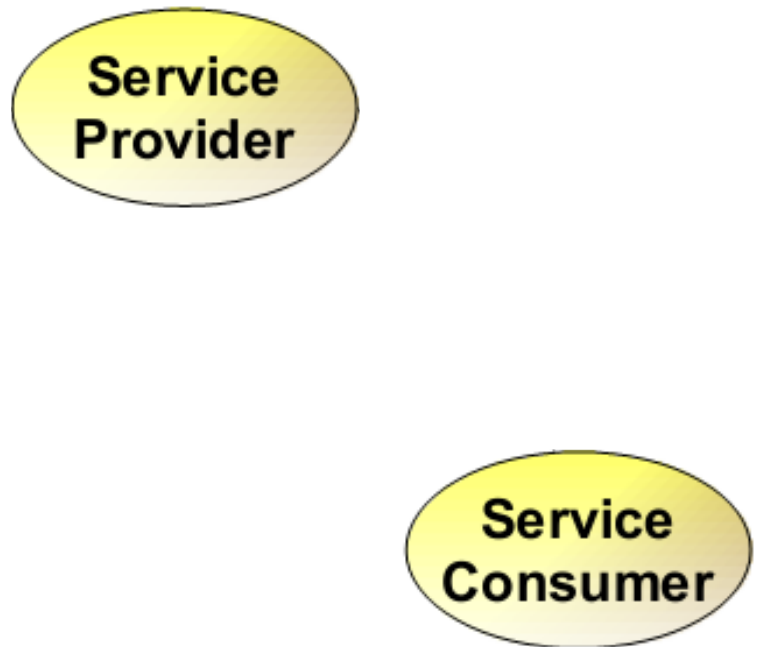
Module 3 Guiding Questions

- What is service discovery?
- What is UDDI?
- What is the relationship between XML, SOAP and UDDI?
- What are the technical aspects of UDDI?
- What are the main uses of UDDI?
- Can you explain the UDDI data model in details?
- How to search UDDI via web based interface?
- How to use the UDDI programmatic API?
- How to publish new companies and services to UDDI?

Module 3 Guiding Questions

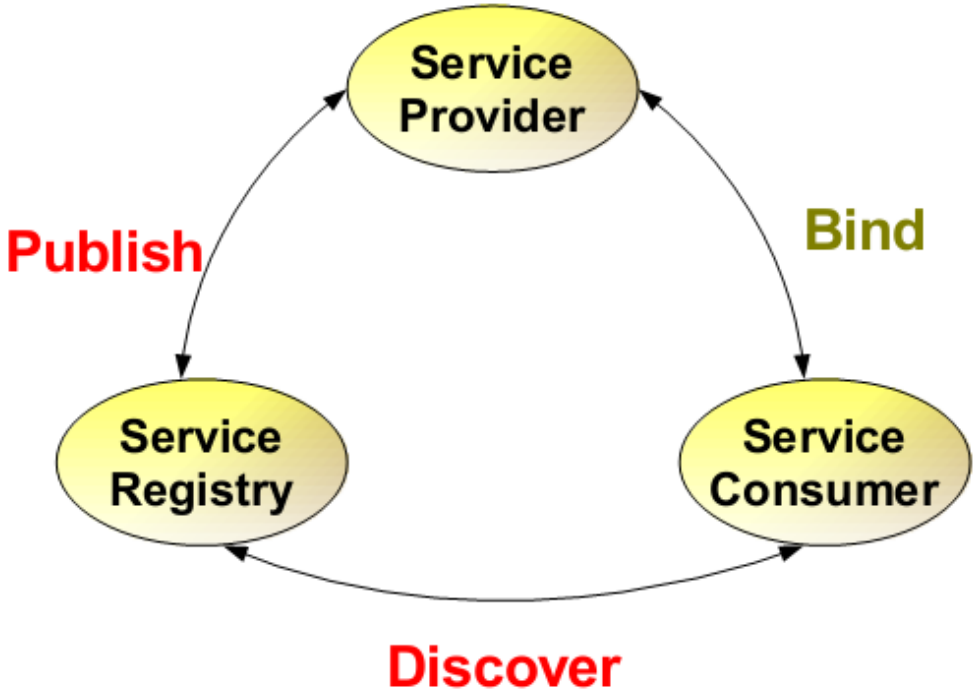
- What is service composition?
- what is business process?
- What is BPEL?
- How to create the business process in BPEL?
- What is the basic structure of BPEL document?

Service Discovery UDDI

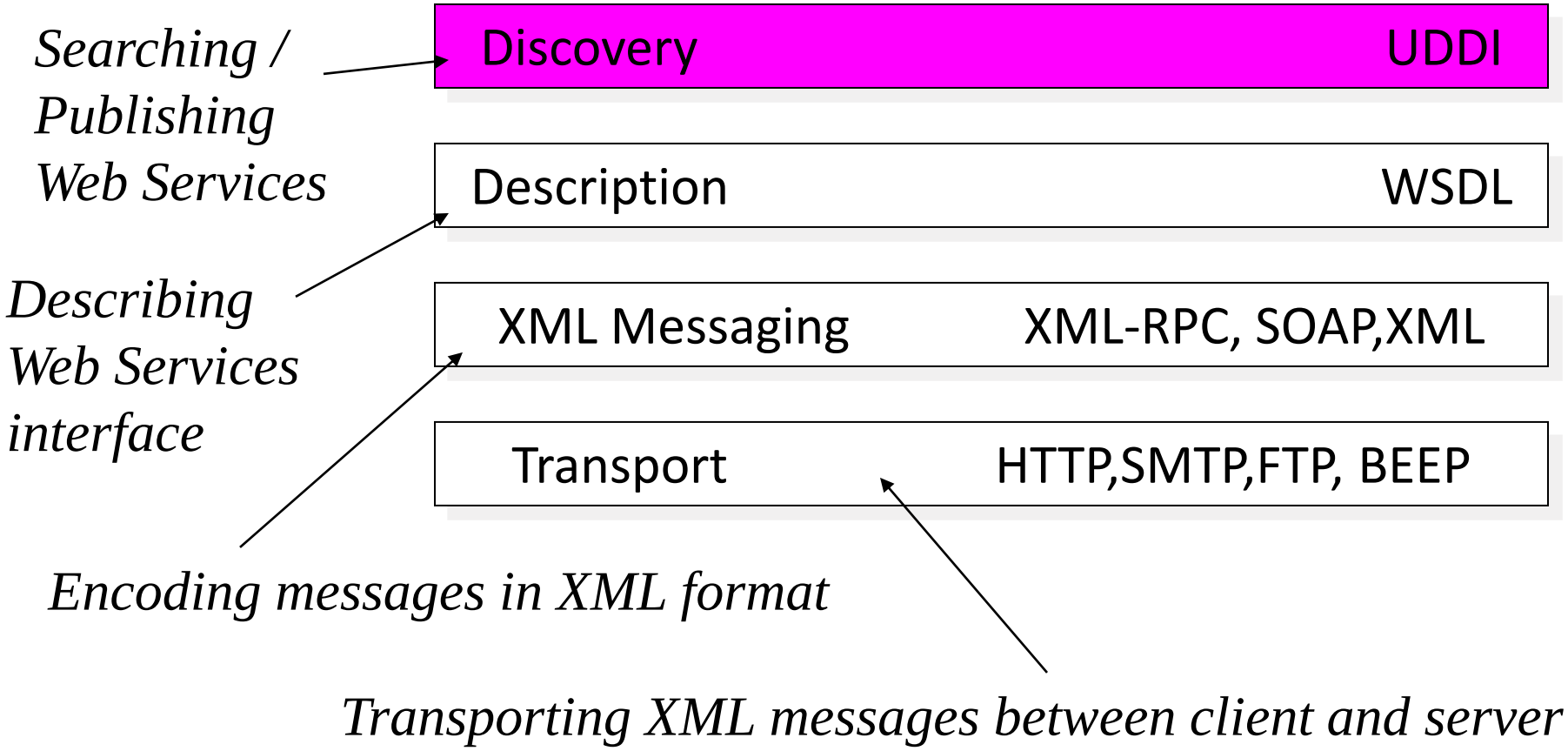

Service
Provider
Service
Consumer

Service Architecture

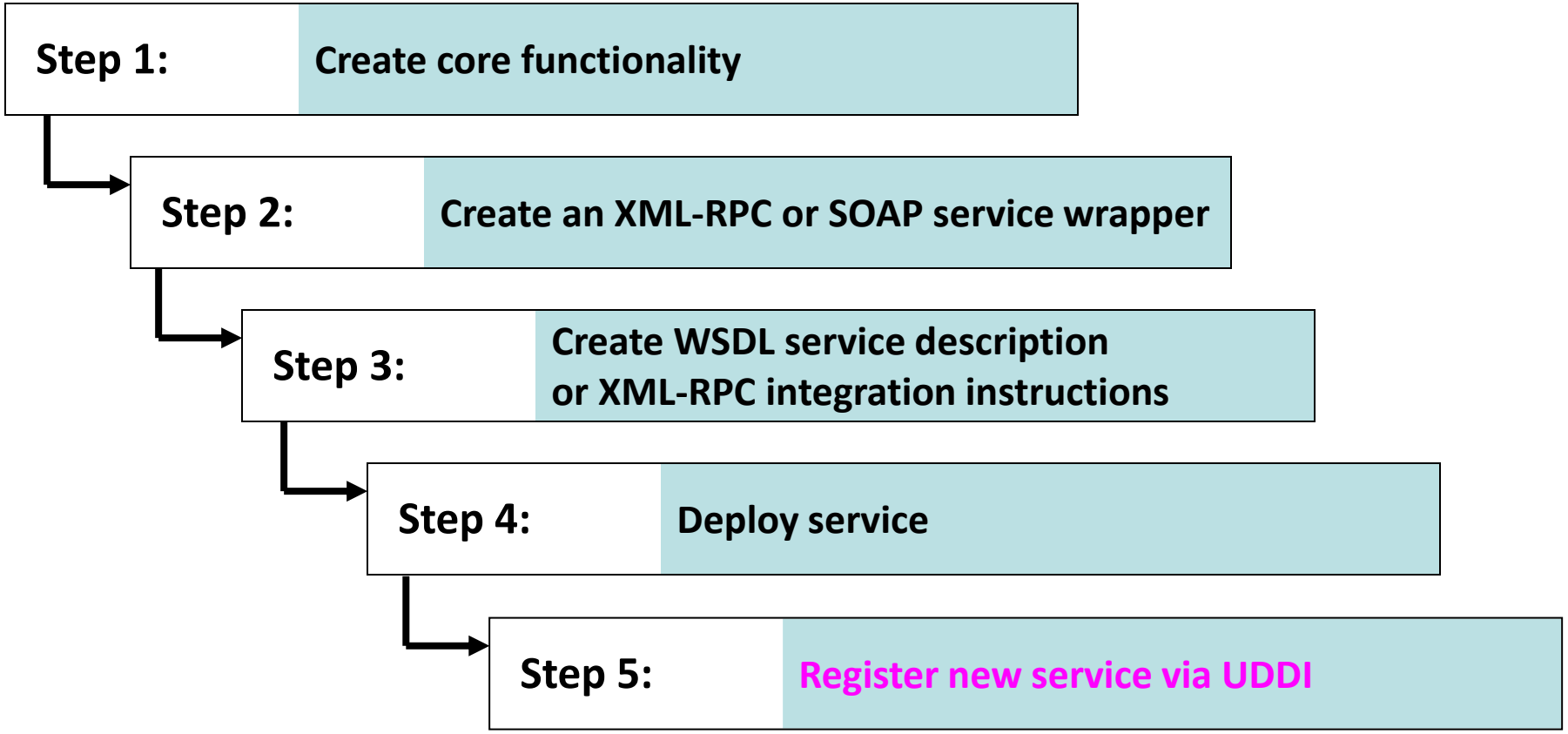
- UDDI(Universal Description, Discovery, and Integration) defines a scheme to publish and discover information about Web services



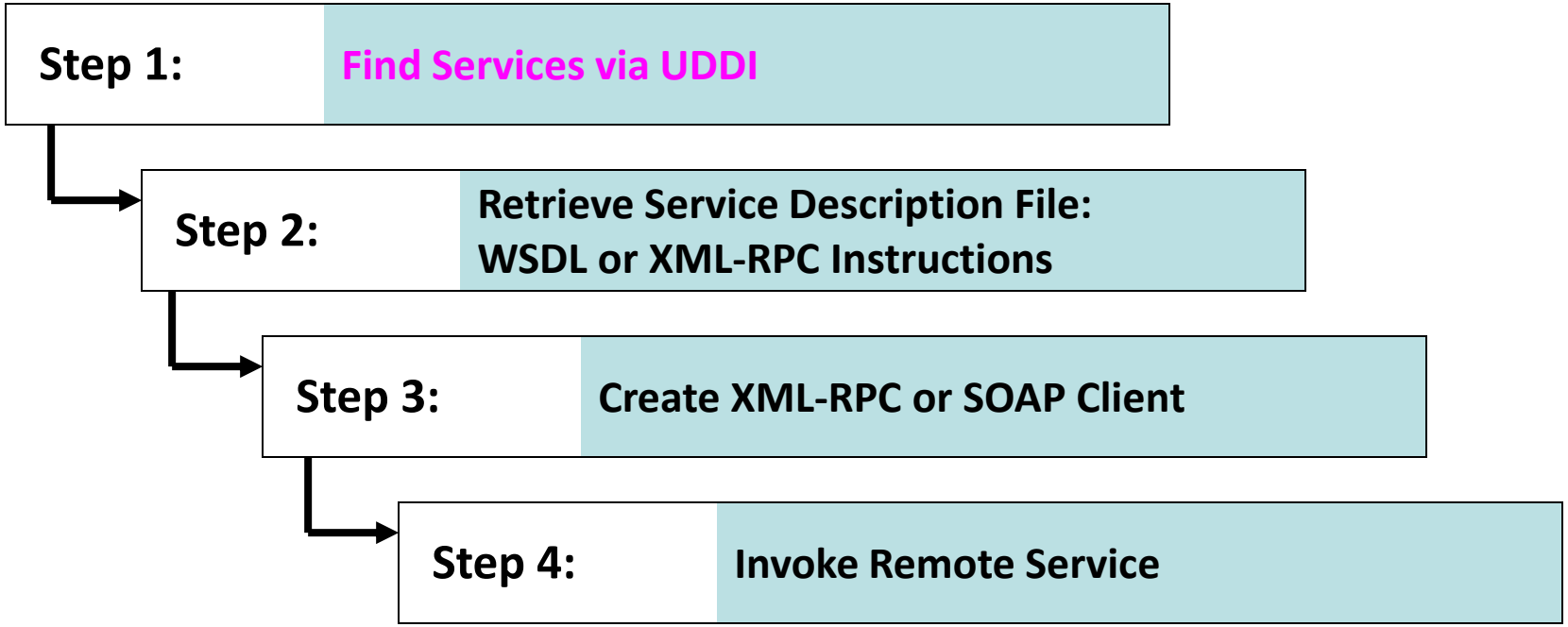
Web Service Protocol Stack



Using the Protocols Together – service provider perspective



Using the Protocols Together – service request perspective



Service discovery

- Automatic detection of devices and services offered by devices on a computer network

Service discovery

- Automatic detection of devices and services offered by these devices on a computer network
- Requires a common language to allow software agents to make use of one another’s services

Service discovery

- Automatic detection of devices and services offered by these devices on a computer network
- Requires a common language to allow software agents to make use of one another’s services
- Web Services Discovery

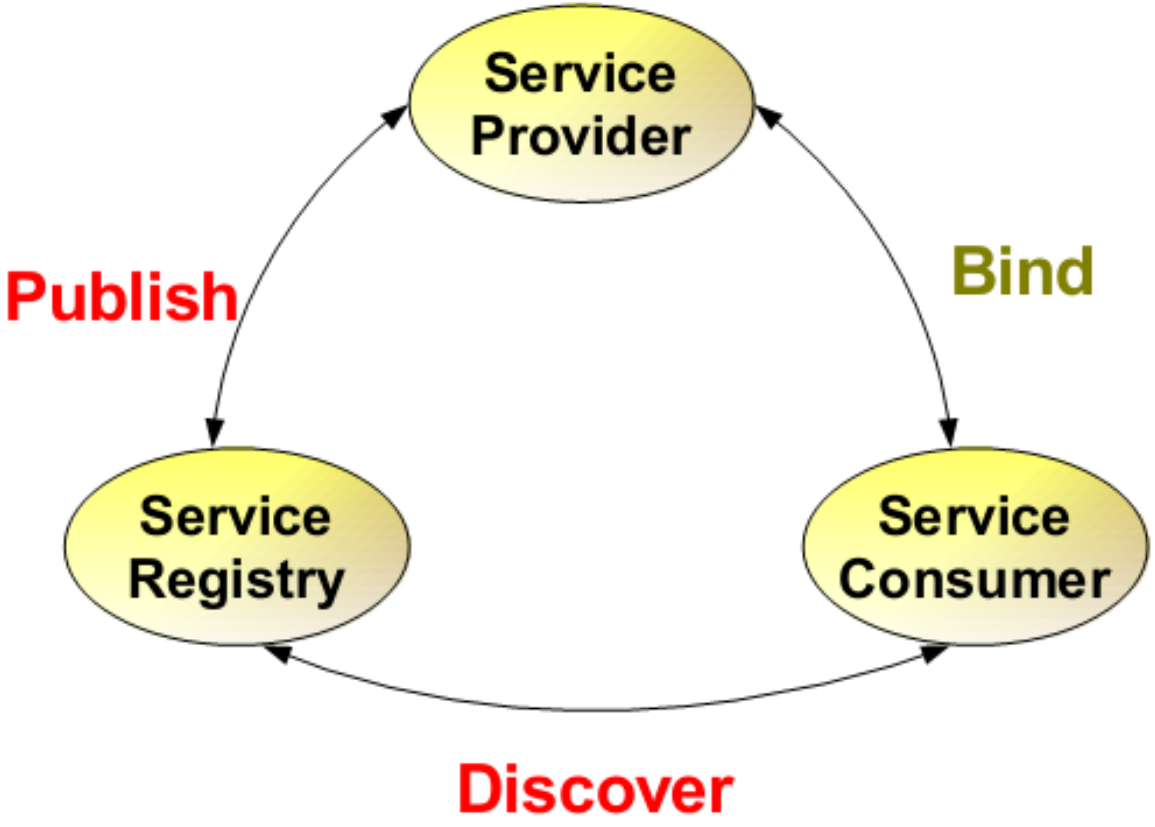
Service discovery

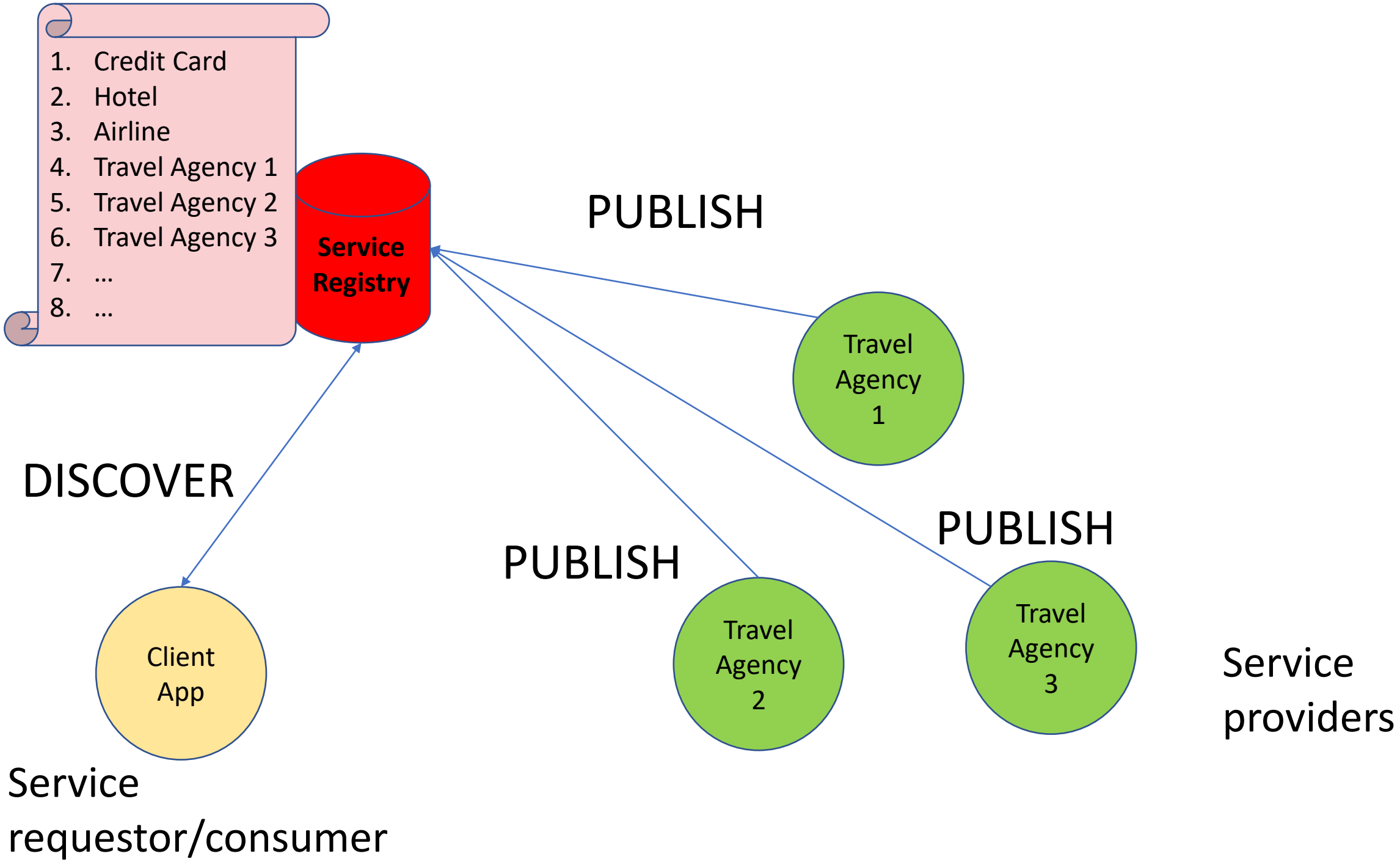
- Automatic detection of devices and services offered by these devices on a computer network
- Requires a common language to allow software agents to make use of one another’s services
- Web Services Discovery
  - provides access to software systems over the Internet using standard protocols

Service discovery

- Automatic detection of devices and services offered by these devices on a computer network
- Requires a common language to allow software agents to make use of one another’s services
- Web Services Discovery
  - provides access to software systems over the Internet using standard protocols
  - the process of finding suitable web services to a given task

Service discovery in Web Service Architecture




1. Credit Card
2. Hotel
3. Airline
4. Travel Agency 1
5. Travel Agency 2
6. Travel Agency 3
7. …
8. …
Service Registry
PUBLISH
Travel Agency 1
DISCOVER
PUBLISH
PUBLISH
Client App
Travel Agency 2
Travel Agency 3
Service providers
Service requestor/consumer

What is UDDI?

- A project to speed interoperability and adoption for web services

What is UDDI?

- A project to speed interoperability and adoption for web services
  - Standards-based specifications for service description and discovery

What is UDDI?

- A project to speed interoperability and adoption for web services
  - Standards-based specifications for service description and discovery
  - Shared operation of a business registry on the web

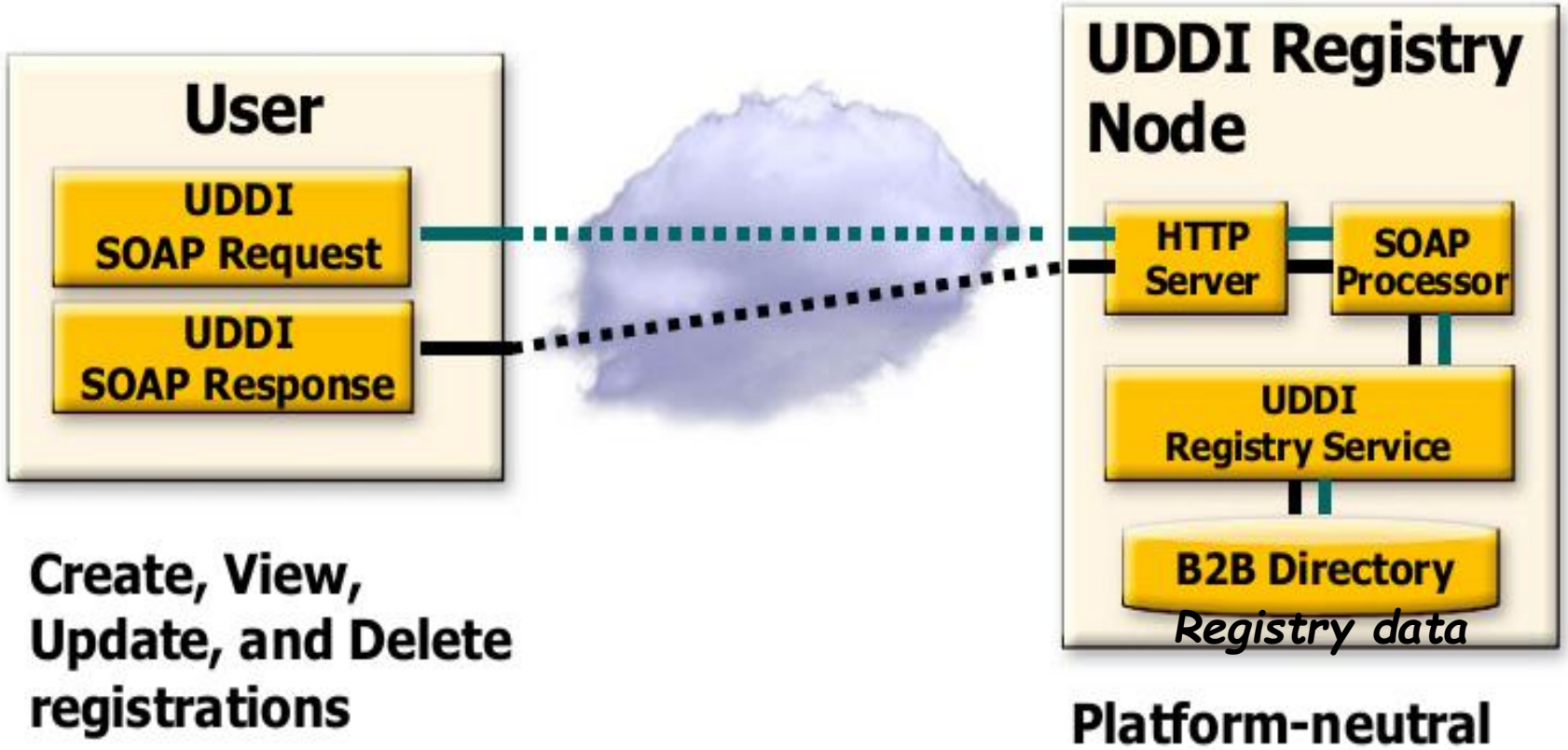
What is UDDI?

- A project to speed interoperability and adoption for web services
  - Standards-based specifications for service description and discovery
  - Shared operation of a business registry on the web
- Partnership among industry and business leaders

What is UDDI?

- Programmatic registration and discovery of business entities and their Web services
- Based on SOAP, HTTP, XML
- Registry data
  - Business registrations
  - Service type definitions

UDDI Runs “Over” SOAP



Why UDDI or something like UDDI?

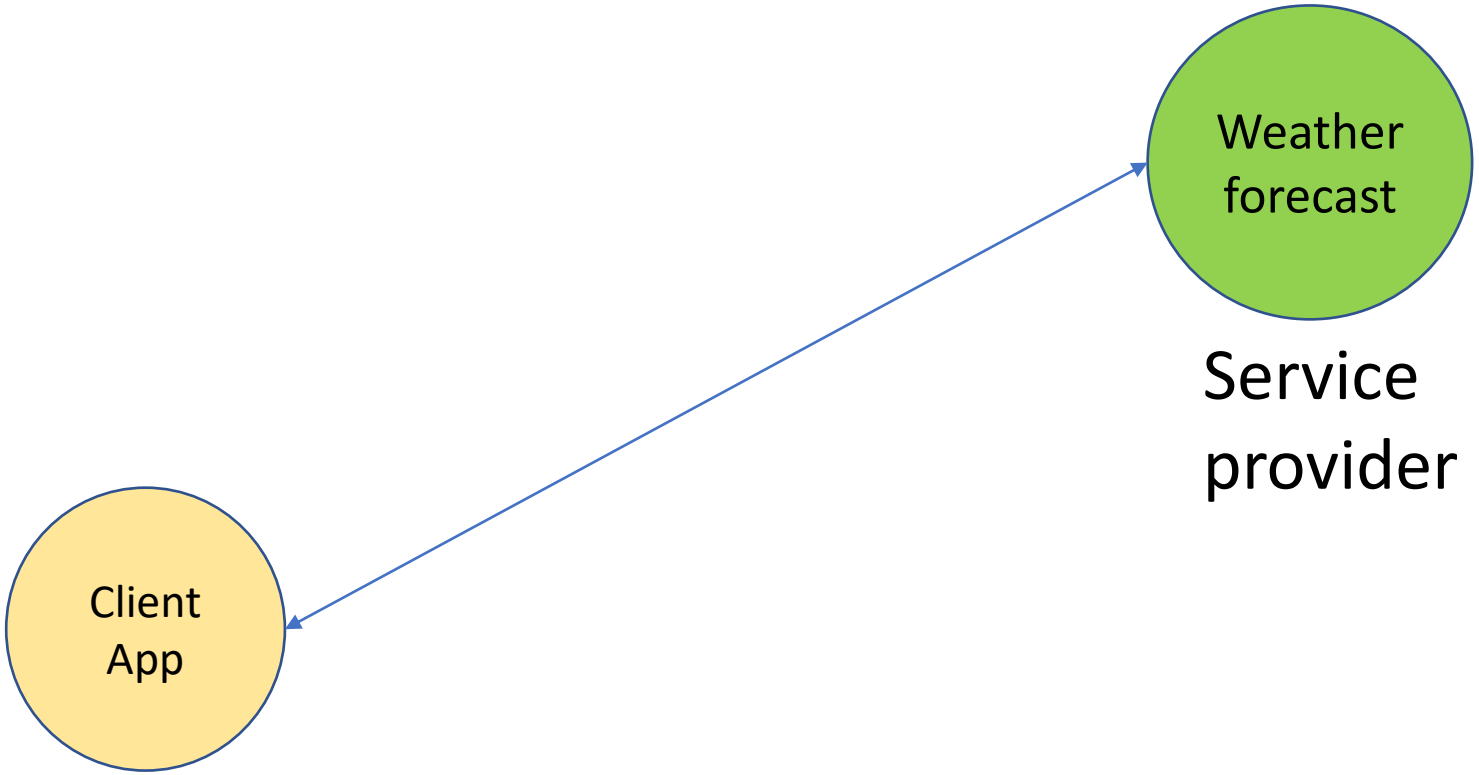
- Platform independent service
  - publication and discovery

Why UDDI or something like UDDI?

- Platform independent service
  - publication and discovery

- Enables dynamic service discovery

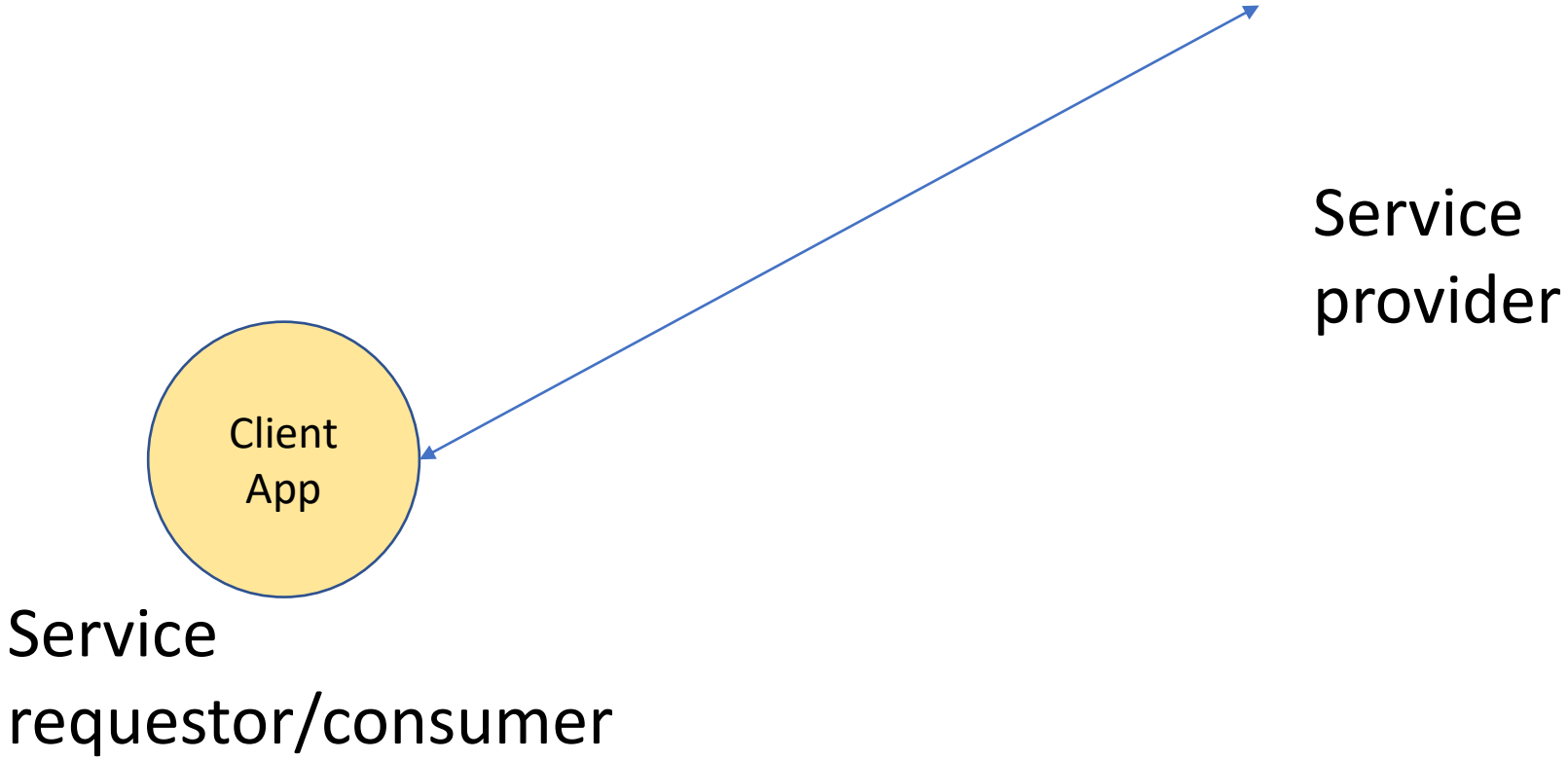
Dynamic service discovery



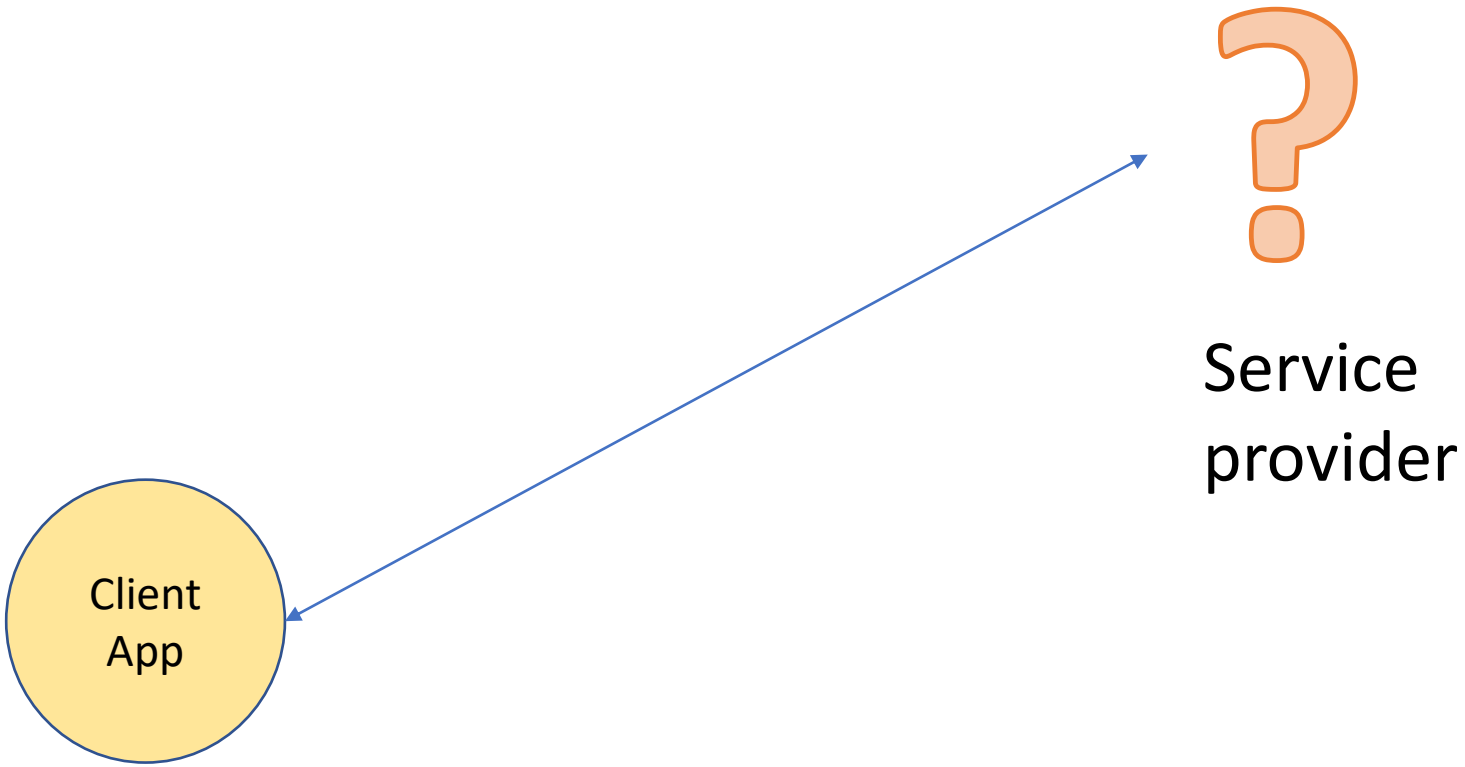
Service provider

Service requestor/consumer

Dynamic service discovery

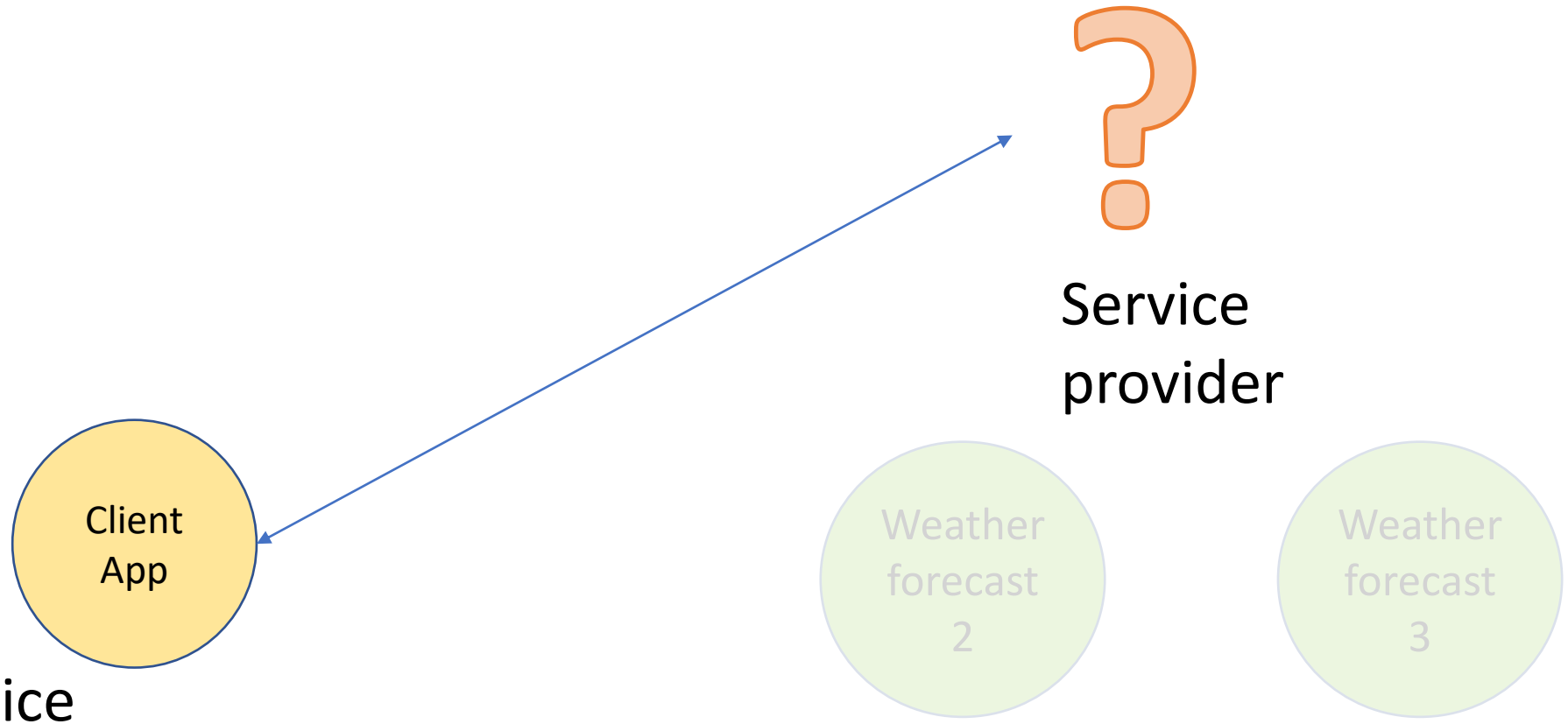


Dynamic service discovery



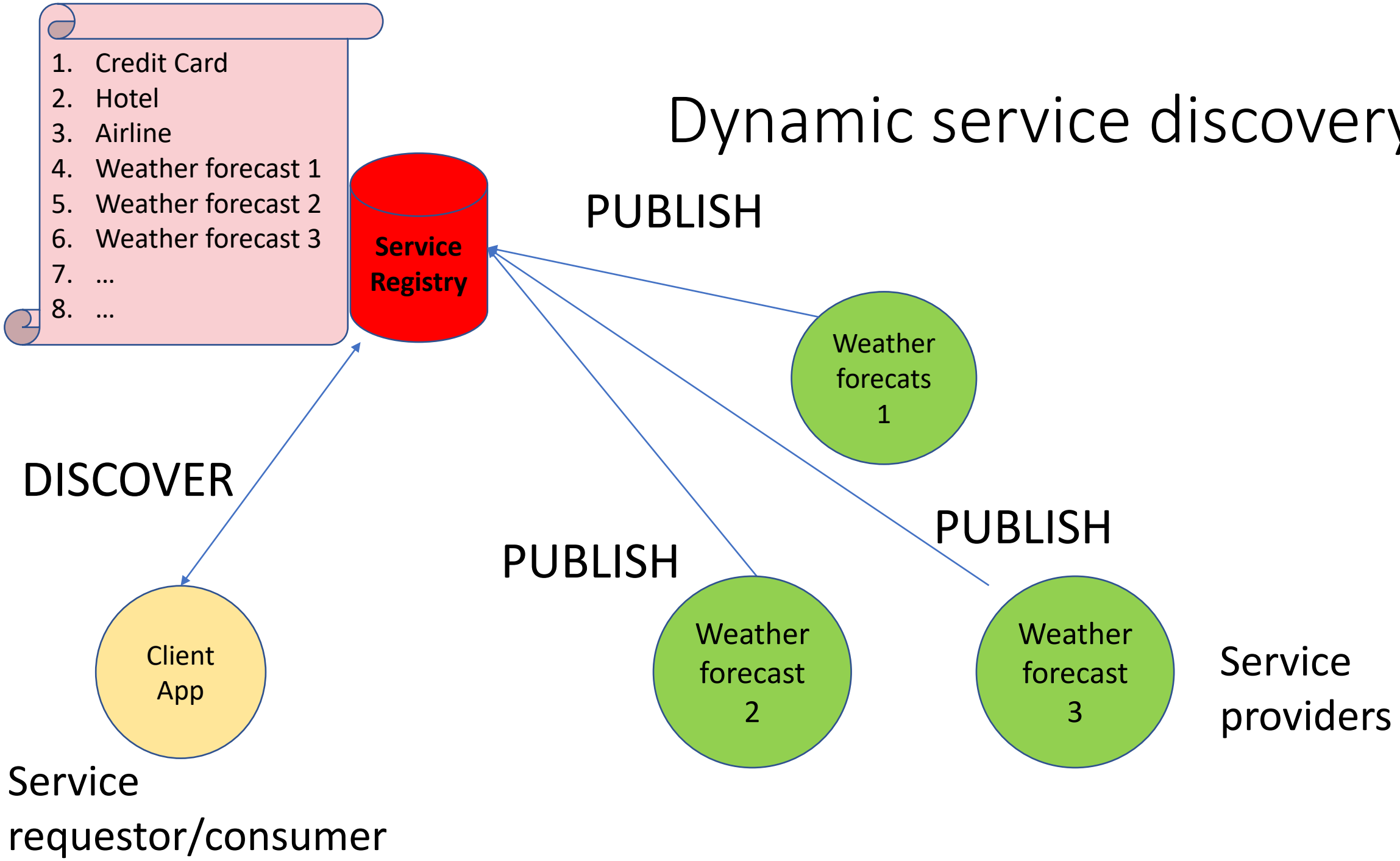
Service
requestor/consumer

Dynamic service discovery



Service
requestor/consumer

Dynamic service discovery



UDDI Vision - 2000

- Open industry initiative, enabling businesses to discover each other and define how they interact over the Internet

UDDI Vision - 2000

- Open industry initiative, enabling businesses to discover each other and define how they interact over the Internet
- Global e-commerce driven by dynamically emerging business relations

UDDI Vision - 2000

- Open industry initiative, enabling businesses to discover each other and define how they interact over the Internet
- Global e-commerce driven by dynamically emerging business relations
- Consumers of web services would be linked up with providers through a public or private dynamic brokerage system

UDDI Vision - 2000

- Open industry initiative, enabling businesses to discover each other and define how they interact over the Internet
- Global e-commerce driven by dynamically emerging business relations
- Consumers of web services would be linked up with providers through a public or private dynamic brokerage system
  - Anyone needing a service would go to their service broker and select a service supporting the desired SOAP service interface, and meeting other criteria

UDDI Vision - 2000

- Open industry initiative, enabling businesses to discover each other and define how they interact over the Internet
- Global e-commerce driven by dynamically emerging business relations
- Consumers of web services would be linked up with providers through a public or private dynamic brokerage system
  - Anyone needing a service would go to their service broker and select a service supporting the desired SOAP service interface, and meeting other criteria
  - The publicly operated UDDI node or broker would be critical for everyone

UDDI Vision - 2000

- Open industry initiative, enabling businesses to discover each other and define how they interact over the Internet
- Global e-commerce driven by dynamically emerging business relations
- Consumers of web services would be linked up with providers through a public or private dynamic brokerage system
  - Anyone needing a service would go to their service broker and select a service supporting the desired SOAP service interface, and meeting other criteria
  - The publicly operated UDDI node or broker would be critical for everyone
    - For the consumer – public or open brokers would only return services listed for public discovery by others

UDDI Vision - 2000

- Open industry initiative, enabling businesses to discover each other and define how they interact over the Internet
- Global e-commerce driven by dynamically emerging business relations
- Consumers of web services would be linked up with providers through a public or private dynamic brokerage system
  - Anyone needing a service would go to their service broker and select a service supporting the desired SOAP service interface, and meeting other criteria
  - The publicly operated UDDI node or broker would be critical for everyone
    - For the consumer – public or open brokers would only return services listed for public discovery by others
    - For service producer – matadata of index categories would be critical for effective placement

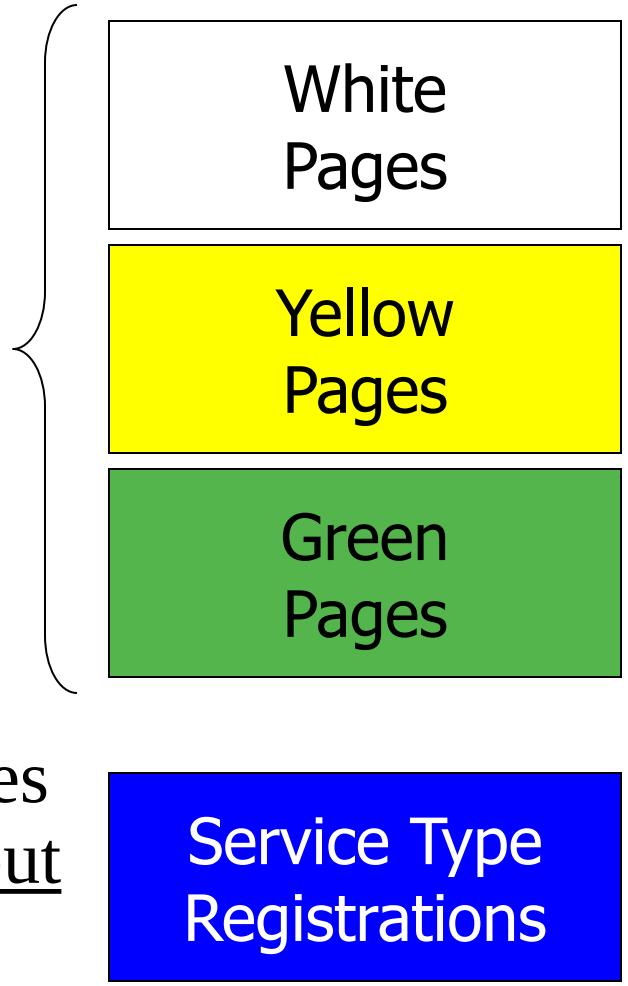
Registry Data

Created by businesses

Created by standard organizations, industry consortium

Registry Data

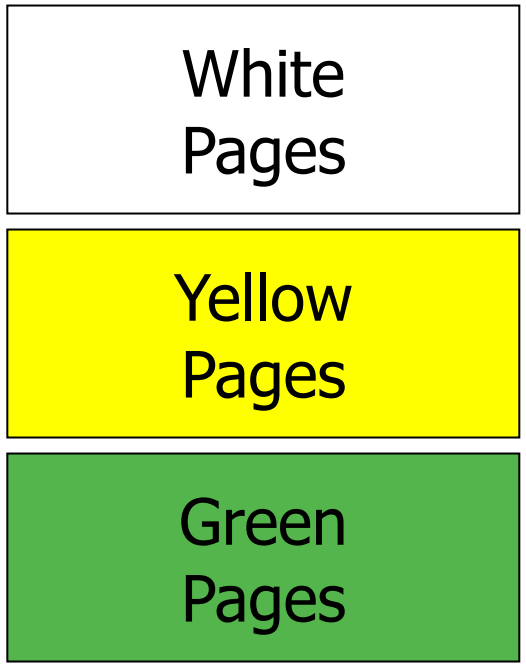
- Businesses register public information about themselves



- Standards bodies, Programmers, Businesses register <u>information about their Service Types</u>

Business Registration Data

- "White pages"
  - Business name, address, contact, and known identifiers
- "Yellow pages"
  - industrial categorizations
    - Industry: NAICS (Industry codes - US Govt.)
    - Product/Services: UN/SPSC (ECMA)
    - Location: Geographical taxonomy
- "Green pages"
  - technical information about services
  - a pointer to an external specification and an address for invoking the web service



What uses UDDI?

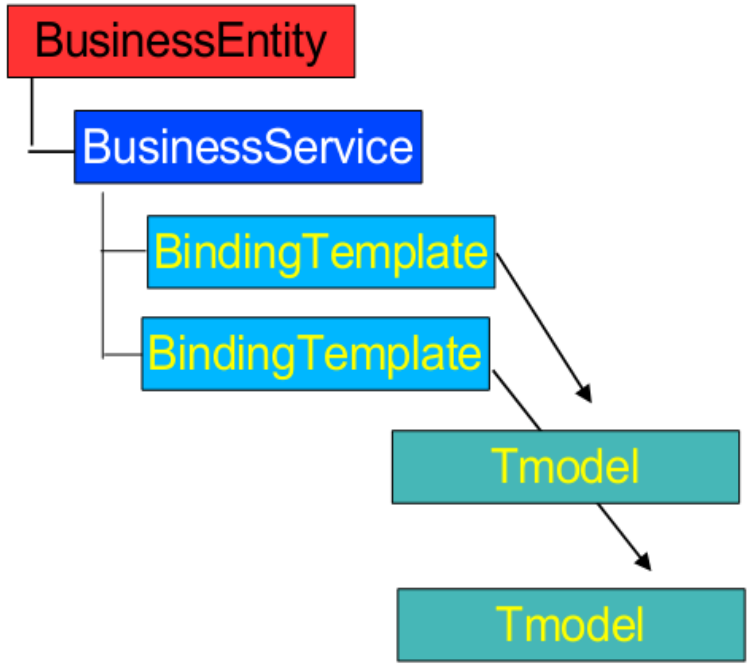
- Tool building client (Service Consumer)
  - Browse or search registry
  - Create a service proxy
- Tool publishing the service
  - Generates WSDL
  - Construct UDDI entries
- Application that needs dynamic binding
  - Directly access UDDI
  - Query can be pre-generated

UDDI Adoption Phases

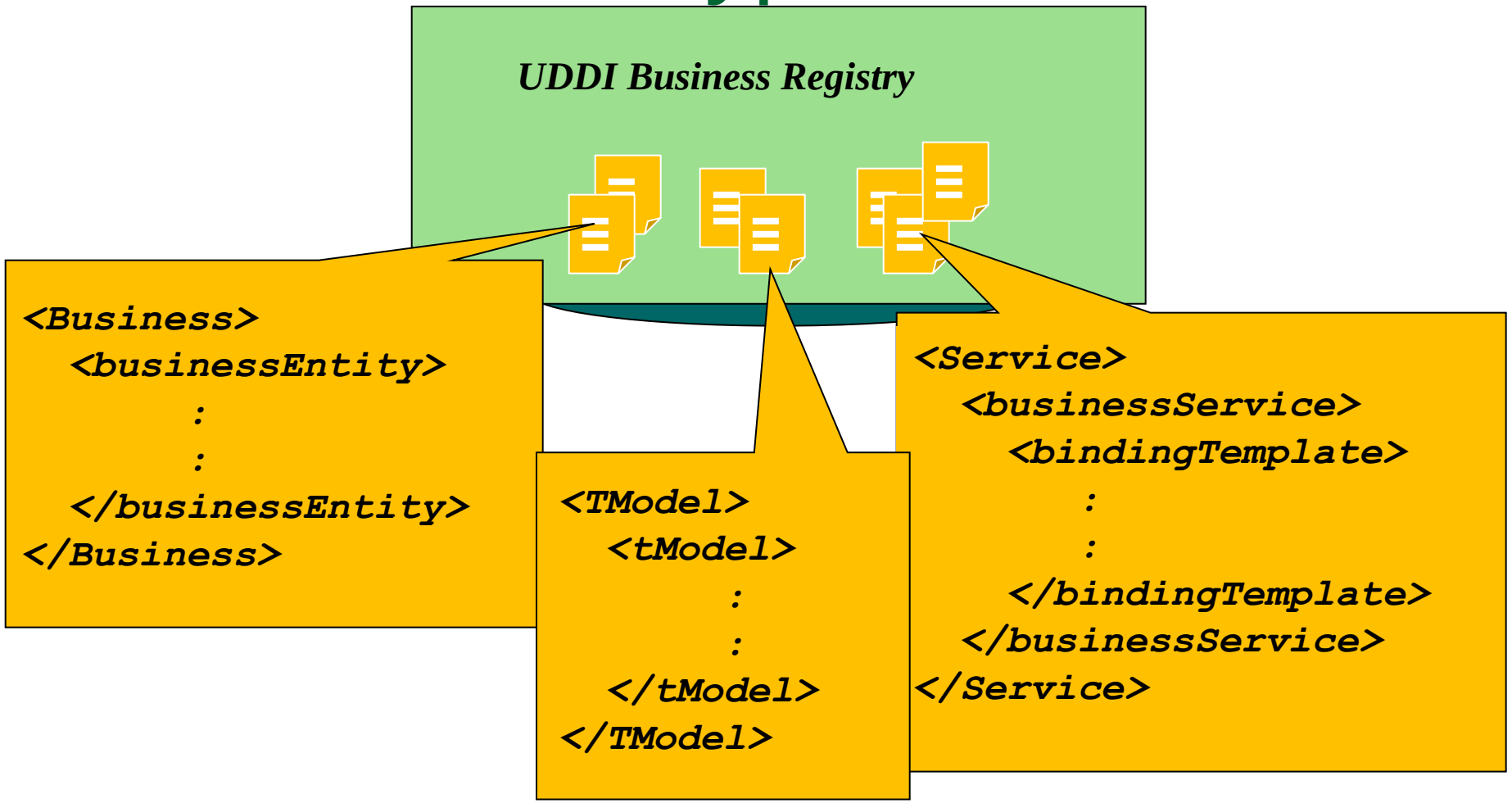
- Phase 1: Experimental stage
- Phase 2: Private UDDI registry within an intranet
- Phase 3: Public UDDI registries with no coordination among them
- Phase 4: Public UDDI registries with coordination (i.e. replication)
- Phase 5: Value added registry services

UDDI Data Model/Types

- UDDI includes an XML Schema that describes four core types of information:
  - businessEntity
    - About the actual business, e.g. business name, etc.
  - businessService
    - About the services provided by the business
  - bindingTemplate
    - About how and where to access a specific service
  - tModel (Technical Model)
    - Include descriptions and pointers to external technical specifications or taxonomies



UDDI Data Model/Types



XML Schema describes these four core types of information

A. businessEntity

```
<businessEntity
  businessKey=
     "ba744ed0-3aaf-11d5-80dc-002035229c64">
  <name> XMethods </name>
  <description> … </description>
  <contacts>
    <contact> … </contact>
    <contact> … </contact>
  </contacts>
  <identifierBag> … </identifierBag>
  <categoryBag> … </categoryBag>
</businessEntity>
```

Typical contents of businessEntity element

A. businessEntity

- businessEntity element includes info about the actual business
  - Business name, description, contact info such as address, phone, contact person, etc.

- Each business will receive a unique businessKey value when registration to a UDDI server
  - e.g. businessKey of Microsoft in its UDDI server: 0076b468-eb27-42e5-ac09-9955cff462a3

- The key is used to tie a business to its published services

A. businessEntity

```
<businessEntity
  businessKey=
     "ba744ed0-3aaf-11d5-80dc-002035229c64">
  <name> XMethods </name>
  <description> … </description>
  <contacts>
    <contact> … </contact>
    <contact> … </contact>
  </contacts>
  <identifierBag> … </identifierBag>
  <categoryBag> … </categoryBag>
</businessEntity>
```

Typical contents of businessEntity element

A. businessEntity

- Can also include other unique value(s) in identifierBag that identifies the company
  - UDDI supports Dun & Bradstreet D-U-N-S® Numbers and Thomas Registry Supplier IDs
  - e.g. Microsoft's Dun & Bradstreet D-U-N-S® No:

    08-146-6849
- Businesses can also register multiple business categories in categoryBag based on standard taxonomies, e.g.
  - NAICS: The North American Industry Classification System provides industry classification
  - UNSPSC: Universal Standard Products and Service Classification provides product and service classification

A. businessEntity

Examples of identifierBag and categoryBag contents (Microsoft)

```
<identifierBag>
  <keyedReference
    tModelKey=
   "uuid:8609c81e-ee1f-4d5a-b202-3eb13ad01823"
    keyName="D-U-N-S" keyValue="08-146-6849" />
</identifierBag>
<categoryBag>
 <keyedReference
    tModelKey=
   "uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"
    keyName="NAICS: Software Publisher"
    keyValue="51121" />
 </categoryBag>
```

B. businessService

```
<businessService
  serviceKey=
    "d5921160-3e16-11d5-98bf-002035229c64"
  businessKey=
    "ba744ed0-3aaf-11d5-80dc-002035229c64">
  <name>XMethods Delayed Stock Quotes</name>
  <description> … </description>
  <bindingTemplates>
    <bindingTemplate>
      :
    </bindingTemplate>
  </bindingTemplates>
</businessService>
```

To tie the service with the business

Typical contents of businessService element

B. businessService

- businessService element includes info about a single web service or a group of related Web services
- Include the name, description and an optional list of bindingTemplates
- Like businessEnitity, each businessService has a unique service key
- Should specify the businessKey to relate with the business that provides that service

B. businessService

- Represents the business services provided by the *businessEntity*
- Unique key used to represent a service
- Name of the service
- Contains *BindingTemplate* structures

```
<businessService businessKey="..." serviceKey="...">
    <name>StockQuoteService</name>
    <description> (...) </description>
    <bindingTemplates>
        (...)
        <bindingTemplate>
            (...)
            <accessPoint urlType="http">
                http://example.com/stockquote
            </accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo tModelKey="...">
                </tModelInstanceInfo>
            <tModelInstanceDetails>
        </bindingTemplate>
    </bindingTemplates>
</businessService>
```

C. bindingTemplate

```
<bindingTemplate
  serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
  bindingKey="…">
  <description xml:lang="en">
      :
  </description>
  <accessPoint URLType="http">
    http://services.xmethods.net:80/soap
  </accessPoint>
  <tModelInstanceDetails>
      :
  </tModelInstanceDetails>
</bindingTemplate>
```

Typical contents of bindingTemplate element

C. bindingTemplate

- **Specifies Network endpoint address**
- **Contains a reference to a tModel**

```
<businessService businessKey="..." serviceKey="...">
    <name>StockQuoteService</name>
    <description> (...) </description>
    <bindingTemplates>
        (...)
        <bindingTemplate>
            (...)
            <accessPoint urlType="http">
                http://example.com/stockquote
            </accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo tModelKey="...">
                </tModelInstanceInfo>
            <tModelInstanceDetails>
        </bindingTemplate>
    </bindingTemplates>
</businessService>
```

UDDI binding options

Name	Description	UUID	Details
uddi-org:smtp	Email-based service	uuid:93335D49-3EFB-48A0-ACEA-EA102B60DDC6	Identifies a service that is invoked via SMTP email. For example, this could specify a person's email address or an SMTP-based SOAP service.
uddi-org:fax	Fax-based service	uuid:1A2B00BE-6E2C-42F5-875B-56F32686E0E7	Identifies a service that is invoked via fax transmissions.
uddi-org:ftp	FTP-based service	uuid:1A2B00BE-6E2C-42F5-875B-56F32686E0E7	Identifies a service that is invoked via FTP.
uddi-org:telephone	Telephone-based service	uuid:38E12427-5536-4260-A6F9-B5B530E63A07	Identifies a service that is invoked via a telephone call. This could include interaction by voice and/or touch-tone.
uddi-org:http	HTTP-based service	uuid:68DE9E80-AD09-469D-8A37-088422BFBC36	Identifies a web service that is invoked via the HTTP protocol. This could reference a simple web page or a more complex HTTP-based SOAP application.

D. tModel

- tModels are primarily used to provide pointers to external technical specifications (e.g wsdl)

- bindingTemplate only provides info about where to access the SOAP binding, but not how to interface with it
- tModel element fills this gap by providing a pointer to an external specification, such as WSDL

- In fact, tModels are not reserved to Web services
- tModels are used whenever it is necessary to point to any external specification, such as the D-U-N-S® no.

Service Type Registration

- Pointer to the namespace where service type is described
  - What programmers read to understand how to use the service
- Identifier for who published the service
- Identifier for the service type registration
  - called a tModelKey
  - Used as a signature by web sites that implement those services

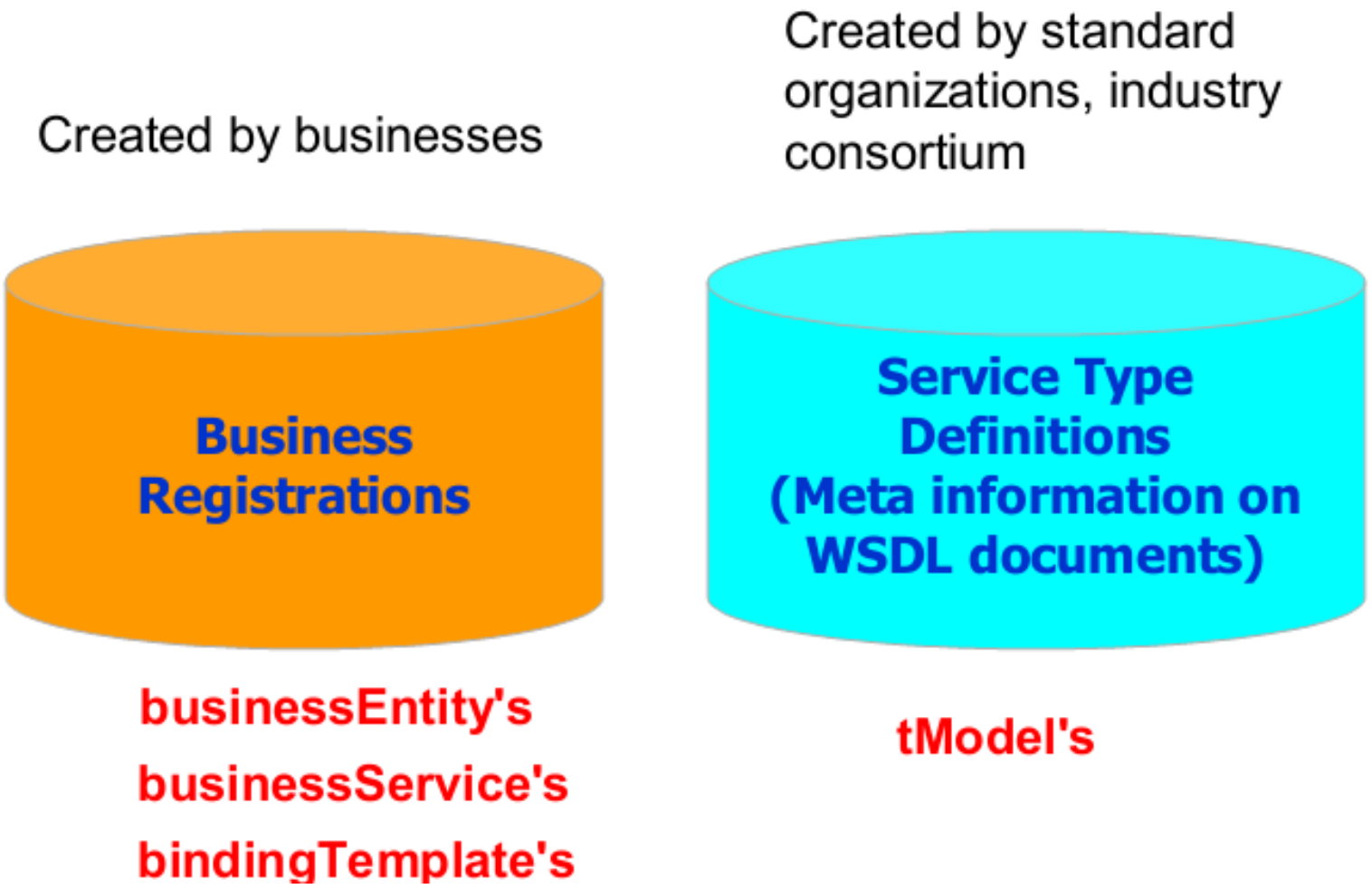
tModel Example

```
<tModel authorizedName="..." operator="..." tModelKey="...">
   <name>StockQuote Service</name>
   <description xml:lang="en">
        WSDL description of a standard stock quote service interface
   </description>
   <overviewDoc>
      <description xml:lang="en"> WSDL source document. </description>
      <overviewURL> http://stockquote-definitions/stq.wsdl  </overviewURL>
   </overviewDoc>
   <categoryBag>
      <keyedReference tModelKey="UUID:..."
                      keyName="uddi-org:types"
                      keyValue="wsdlSpec"/>
   </categoryBag>
</tModel>
```

categoryBag Element

- Allows businessEntity, businessService and tModel structures to be categorized according to any of several available taxonomy based classification scheme
  - NAICS (Industry code)
  - UNSPAC
  - D-U-N-S
  - ISO 3166
  - SIC

Registry Data



Publishing Services

- Publishers interface
  - Save things
    - save_business
    - save_service
    - save_binding
    - save_tModel
  - Delete things
    - delete_business
    - delete_service
    - delete_binding
    - delete_tModel
  - security…
    - get_authToken
    - discard_authToken

4 messages to save each of the 4 structures

- Each save message accepts as input the authToken and one or more corresponding structures.

4 messages to delete each of the 4 core structures

- They all accept the corresponding `uuid` key as the parameter.

Security:

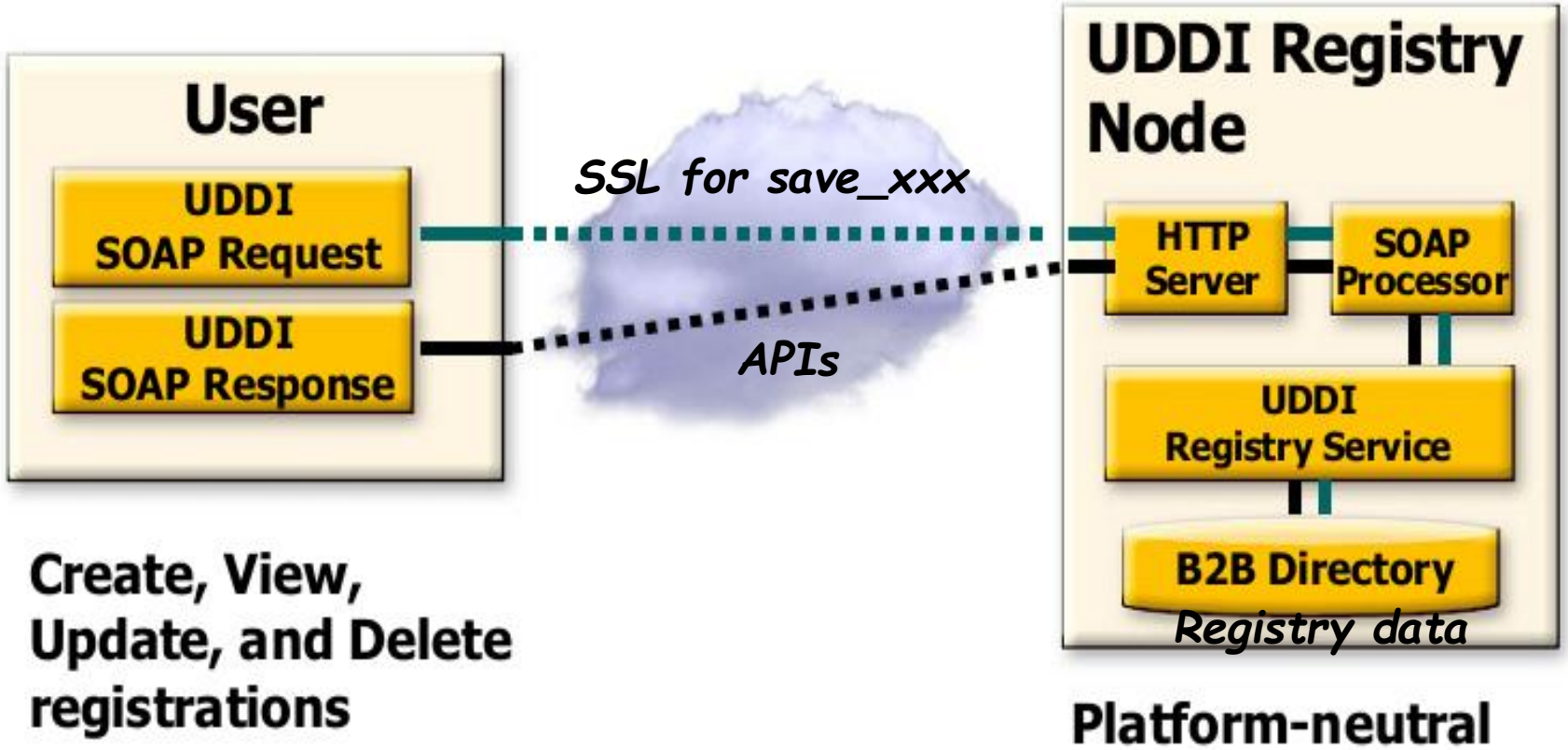
- request an authentication token
- inform registry that the authToken is no longer valid.

Programmer's API: Service Discovery

- Inquiry interface
  - Find things
    - **Find_business**
    - **Find_service**
    - **find_binding**
    - **find_tModel**
  - Get details
    - **Get_businessDetail**
    - **get_serviceDetail**
    - **get_bindingDetail**
    - **Get_tModelDetail**
- Taxonomy interface
    - **validate_categorization**

- Browse
  - 4 messages to **find** each of the 4 structures
- Drill-down
  - The get call can be used to get information regarding a specific instance of any of the 4 data types, given the key

UDDI Runs “Over” SOAP



SOAP Message Example for *get_serviceDetail* request

```
<Envelope>
  <Body>
    <get_serviceDetail generic="1.0">
        <serviceKey>6FD77EF6-E7D6-6FF6-1E41-EBC80107D7B5
        </serviceKey>
    </get_serviceDetail>
  </Body>
</Envelope>
```

SOAP Message Example for get_serviceDetail response


```
<Envelope>
  <Body>
    <serviceDetail generic="1.0" operator="XMethods">
      <businessService serviceKey="6FD77EF6-E7D6-6FF6-1E41-EBC80107D7B5"
                       businessKey="D1387DB1-CA06-24F8-46C4-86B5D895CA26">
        <name>Currency Exchange Rate</name>
        <description>Endpoint for service</description>
        <description>IMPLEMENTATION: glue</description>
        <description>CONTACT EMAIL: support@xmethods.net</description>
        <bindingTemplates>
          <bindingTemplate bindingKey="0036DEBC-2F1B-EB84-09E2-3A4332C3E8B4"
                           serviceKey="6FD77EF6-E7D6-6FF6-1E41-EBC80107D7B5">
            <description>SOAP binding</description>
```

```
<accessPoint
URLType="http">http://services.xmethods.net:80/soap</accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo tModelKey="uuid:D784C184-99B2-
DA25-ED45-3665D11A12E5"/>
            </tModelInstanceDetails>
        </bindingTemplate>
      </bindingTemplates>
    </businessService>
  </serviceDetail>
 </Body>
</Envelope>
```

http://www.ibm.com/developerworks/cn/webservices/ws-uwsdl/part1/

UDDI, WSDL Relationships

WSDL Service Implementation

```
<definitions name="StockQuoteService"
    targetNamespace="http://...">

  <import namespace="http://..."
      location="http://...">

  <service name="StockQuoteService">

    <port name="SingleSymbolService"
      binding="iface:SingleSymbolBinding">
    ...
  </service>
</definitions>
```

UDDI Registry

```
<businessEntity businessKey="...">
  <name>Stock Quote Service, Inc.</name>
  ...

  <businessService serviceKey="..."
    <name>StockQuoteService</name>
    ...
    <bindingTemplates>
      <bindingTemplate bindingKey="...">
        ...
      <tModelInstanceInfo tModelKey="...">
        ...
        <overviewDoc>
          <overviewURL>
            http://.../SQS.wsdl
          </overviewURL>
        ...
    </bindingTemplates>
  </businessService>
</businessEntity>
```

WSDL Service Interface

```
<definitions
    name="StockQuoteService-interface"
    targetNamespace="http://..." >

  <message name="SingleQuoteRequest">
  </message>
    ...
  <portType name="SingleSymbolService">
  </portType>
    ...
  <binding name="SingleSymbolBinding"
           type="tns:SingleSymbolService">
    ...
  </binding>
</definitions>
```

```
<tModel tModelKey="...">
  <name>http://...</name>
  <overviewDoc>
    <overviewURL>
      http://.../SQS-interface.wsdl
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="..."
        keyName="uddi-org:types"
        keyValuee="wsdlSpec"/>
  </categoryBag>
```

Steps that could be Performed by Industry Consortium (for tModel)

- Create WSDL document that contains abstract part of service definition (WSDL interface definition)

- Create tModel that
  - makes a URL reference to WSDL interface definition
  - includes category information
  - can be shared by many business entities

- Register the tModel to UDDI registry

Steps that are performed by Business entities
(for bindingTemplate)

- Find *tModel* for a particular service to offer from the UDDI registry
- Determine the port address
- Create bindingTemplate that
  - contains the port address
  - makes a reference to the previously found tModel
- Create businessService that refers to the bindingTemplate
- Create businessEntity if necessary

Discovery of a Service

- Programmatically
  - via Categorization (Yellow paging)
  - via identity information (White paging)
  - via Drill-down
  - via name patterns
- Through UDDI Browser

Binding to and Invocation of a Service

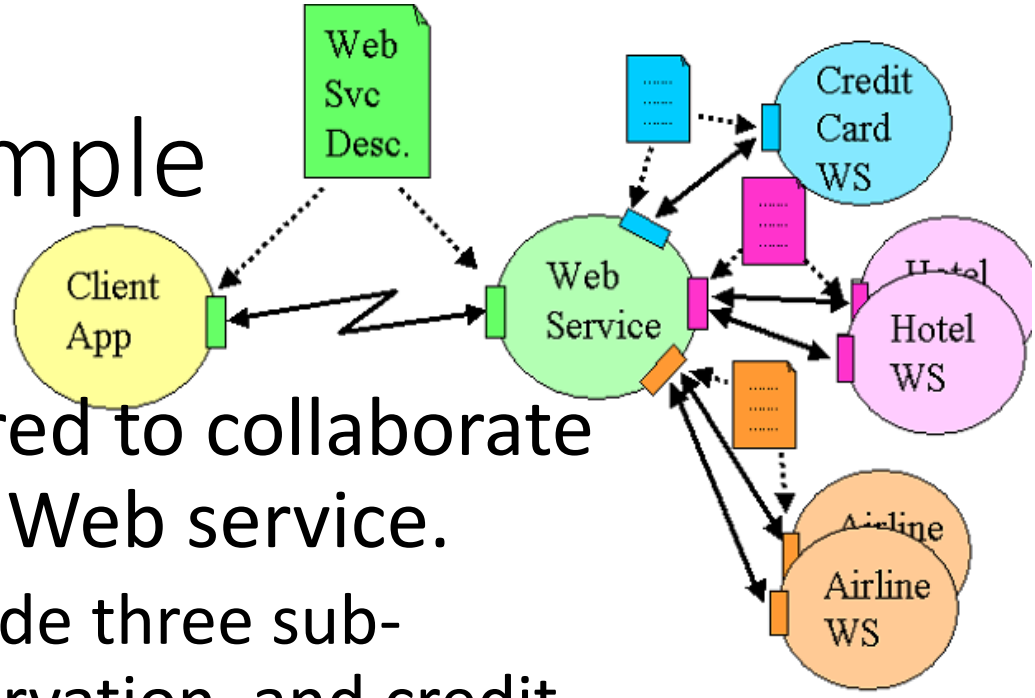
- Obtain WSDL interface information from the *tModel*
- Obtaining port address from *bindingTemplate*
- Construct WSDL **instance** definition (WSDL document with concrete binding and port address)
- Create service proxy from WSDL
- Invocation pattern
  - Cache the bindingTemplate info for a service
  - If call to web service fails, re-check info in UDDI

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



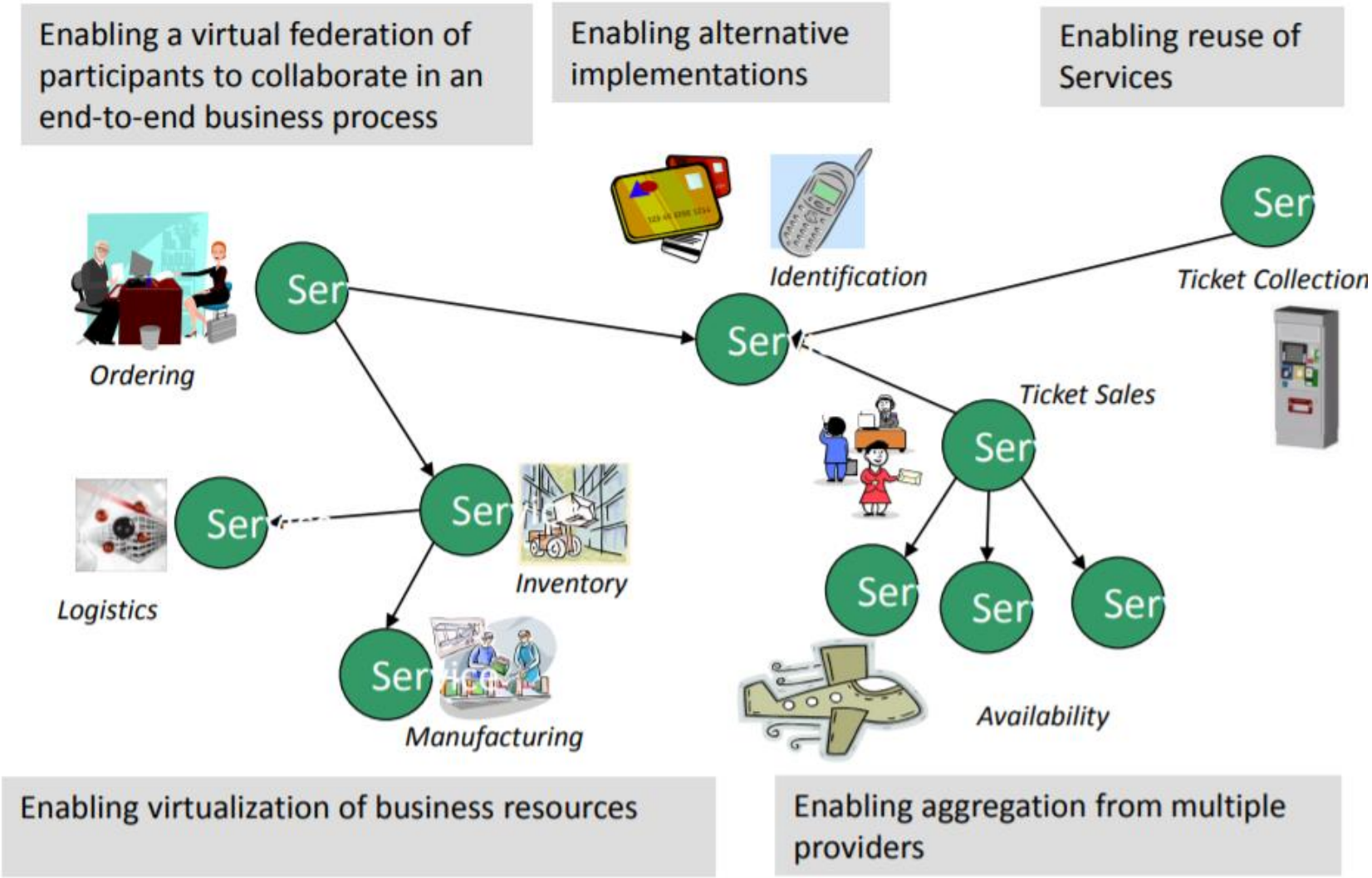
- Multiple Web services may be required to collaborate with each other to form a composite Web service.
  - A travel booking Web service may include three sub-processes: flight reservation, hotel reservation, and credit card payment.
  - These three sub-processes may be performed by three individual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

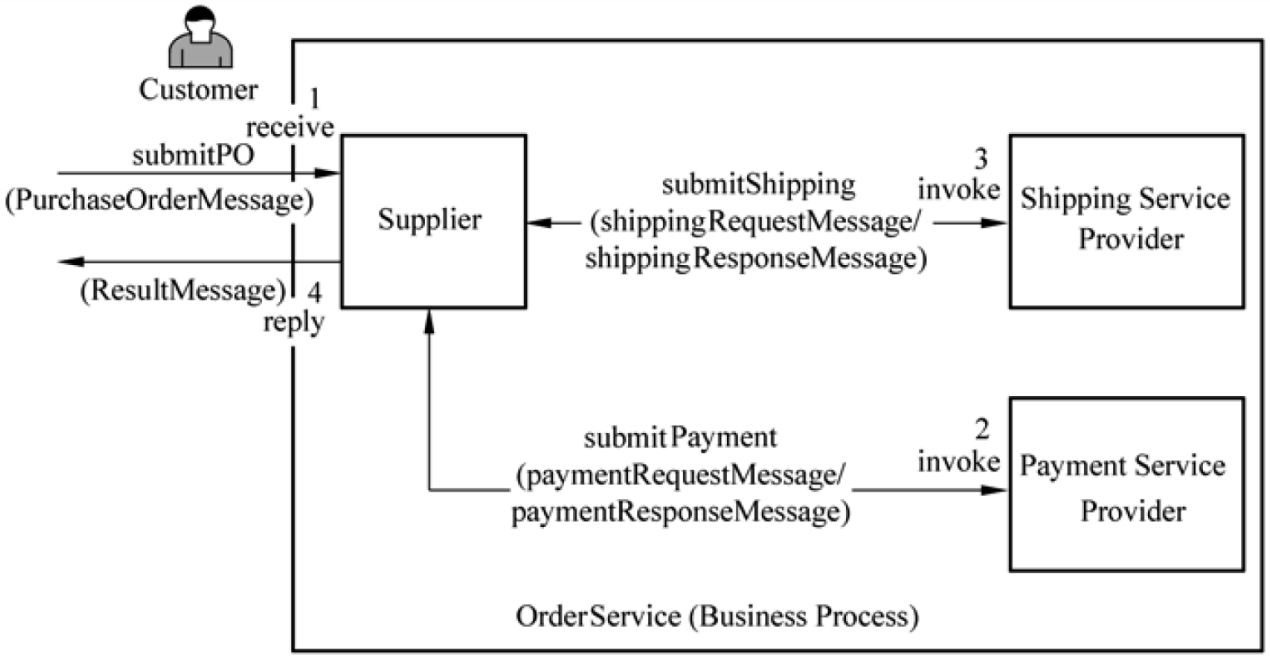


What is BPEL?

- 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>
```

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