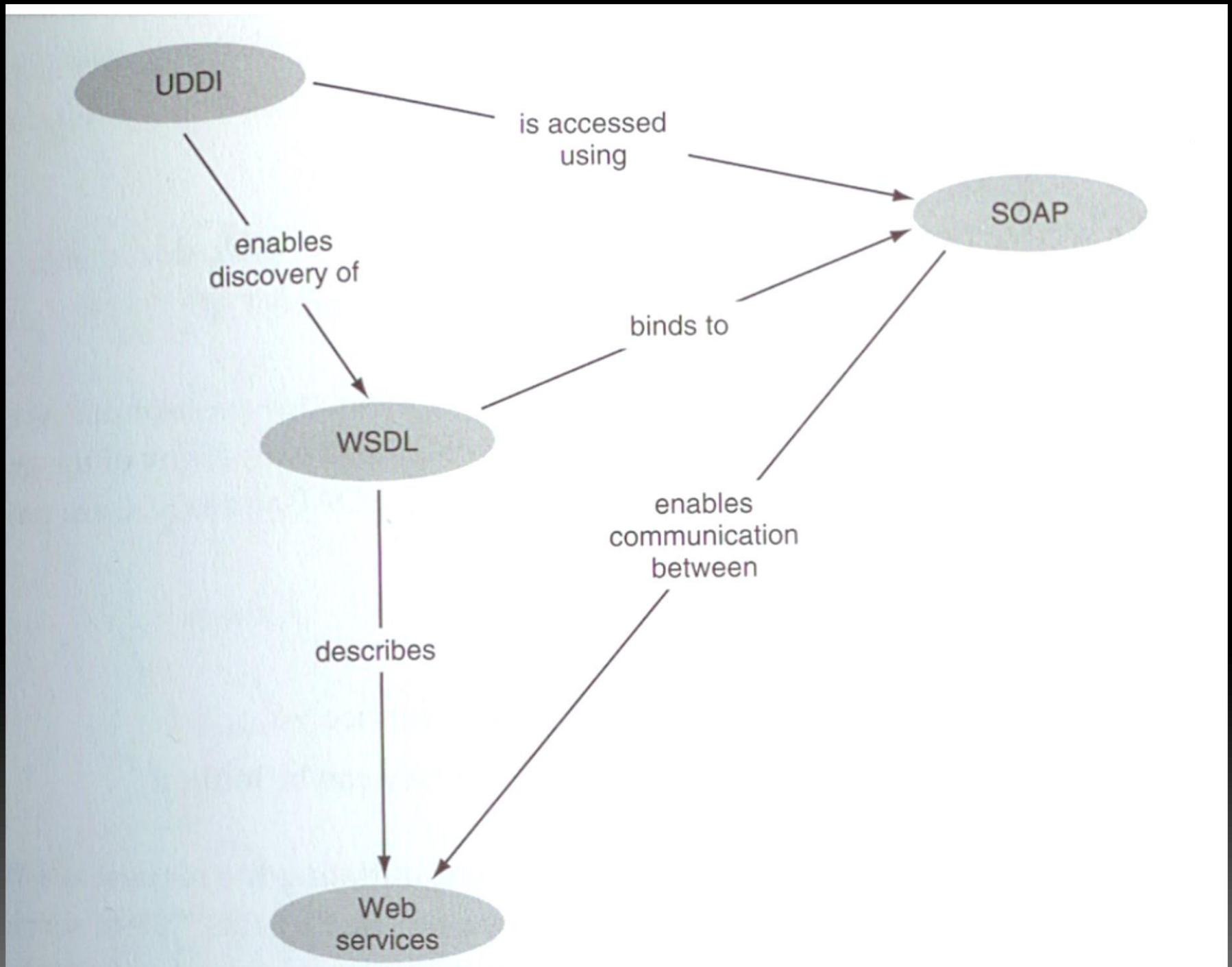
WSDL, SOAP, UDDI

Time Line

- 2000, W3C accepted SOAP
  - Simple Object Access Protocol
  - CORBA, DCOM
- 2001, W3C released WSDL
  - Web Service Description Language
- UDDI
  - Universal Description, Discovery, and Integration

Services

- Like components
  - Independent block for a application for some functions
- Unlike Components
  - Autonomy from others services
  - Responsible to its own domain


UDDI
is accessed using
SOAP
enables discovery of
binds to
WSDL
enables communication between
describes
Web services

XML Web Services

1. Communicates via Internet Protocols
   - Most commonly HTTP
2. Sends/Receives data formatted as XML Document

- You can get yourself an "I'm Service Oriented" T-Shirt.
  - Write a PHP or ASP
  - Sending and receiving data by XML format

However…

- You need to provide a service description
  - At minimum, consists of a WSDL document
- Transporting XML document over HTTP
  - By SOAP
- Be able to act
  - Requestor and provider
- Discovered by an agent
  - Through UDDI

SOA

- What kind of application?
  - An application that uses web services.
  - An application based on a service oriented architecture
- SOA applications follow a set of conventions.

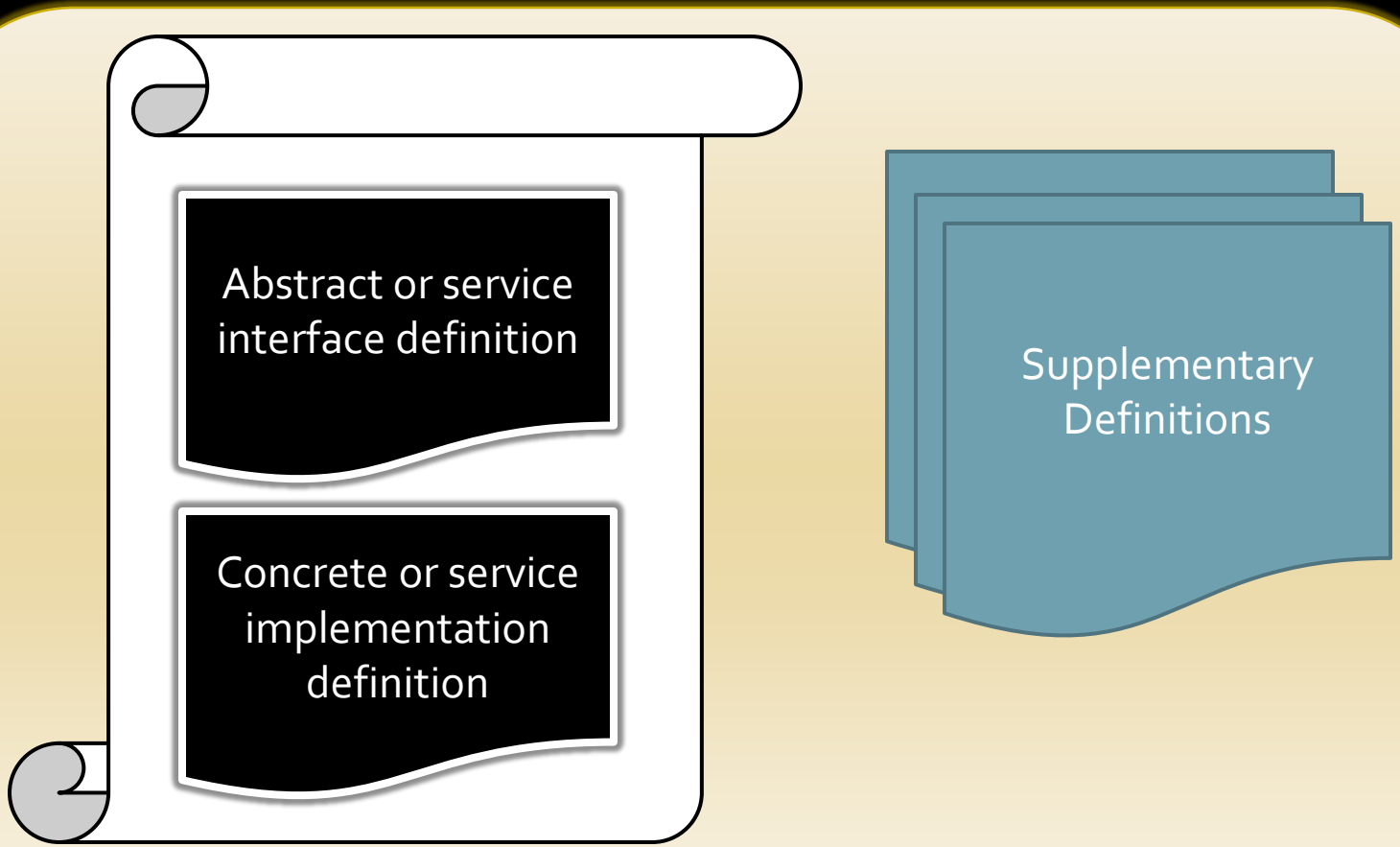
Common Principles of SOA

- Reusable logic is divided into services
- Services share a formal contract
  - Information Exchange
  - Supplementary service description
- Services are loosely coupled
- Services abstract underlying logic
  - Nature or form of the underlying logic is invisible and irrelevant to other services

Common Principles of SOA

- Services are composable
- Services are autonomous
- Services are stateless
  - Services should be designed to maximize statelessness
- Service are discoverable
  - By human beings
  - By other agents or serverices

Web Service Description Structure



Web Service Description Structure

- Service Definition = Abstract + Concrete
- Service Description = Service Definition + Supplementary Definitions

First Generation Web Services

- Web Services Description Language
- Simple Object Access Protocol
- Universal Description, Discovery, and Integration

- XML-Driven Service Oriented Architecture

WSDL

WEB SERVICES DESCRIPTION LANGUAGE

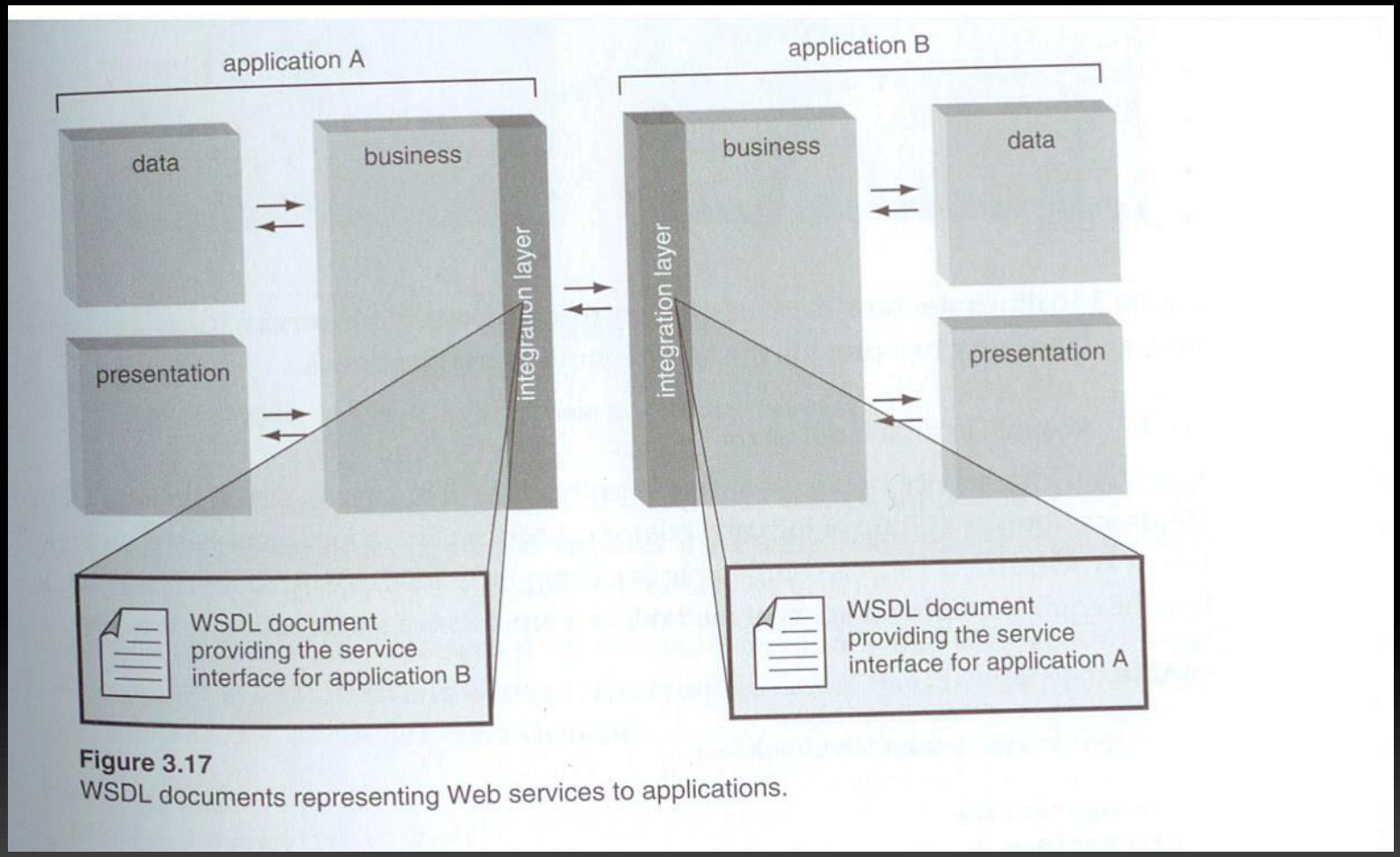


Figure 3.17
WSDL documents representing Web services to applications.

Framework

```
<definitions>
    <types />
    <interface name="xxx">
    …
    </interface>
    <message name="yyy">
    …
    </message>
    <service>
    …
    </service>
    <binding name="zzz">
    …
    </binding>
</definitions>
```

Real Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo"
    xmlns:intf="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types />
<wsdl:message name="echoResponse">
    <wsdl:part name="echoReturn" type="xsd:string" />
 </wsdl:message>
    <wsdl:message name="echoRequest">
    <wsdl:part name="ino" type="xsd:string" />
 </wsdl:message>
<wsdl:portType name="Echo">
    - <wsdl:operation name="echo" parameterOrder="ino">
        <wsdl:input message="impl:echoRequest" name="echoRequest" />
        <wsdl:output message="impl:echoResponse" name="echoResponse" />
    </wsdl:operation>
 </wsdl:portType>
```

```
<wsdl:binding name="EchoSoapBinding" type="impl:Echo">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="echo">
        <wsdlsoap:operation soapAction="" />
        <wsdl:input name="echoRequest">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo"  use="encoded" />
        </wsdl:input>
        <wsdl:output name="echoResponse">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding
                                    namespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo"
use="encoded" />
        </wsdl:output>
    </wsdl:operation>
 </wsdl:binding>
<wsdl:service name="EchoService">
    <wsdl:port binding="impl:EchoSoapBinding" name="Echo">
        <wsdlsoap:address location="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo" />
    </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Did I Write That WSDL by Hand?

- Are you kidding?
- It was automatically generated by axis. Most other Web service tools will do the same.
- See http://grids.ucs.indiana.edu:8045/GCWS/services/Echo?wsdl for generated WSDL.
- We will go through the construction, though, for understanding.

WSDL Parts

- Types
  - Used to define custom message types
- Messages
  - Abstraction of request and response messages that my client and service need to communicate.
- PortTypes
  - Contains a set of operations.
  - Operations organize WSDL messages.
  - Operation->method name, PortType->java interface
- Bindings
  - Binds the PortType to a specific protocol (typically SOAP over http).
  - You can bind one PortType to several different protocols by using more than one port.
- Services
  - Gives you one or more URLs for the service.
  - Go here to execute "echo".

ECHO SERVICE WSDL, SECTION BY SECTION

Front Matters

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
    targetNamespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo"
    xmlns:intf="http://grids.ucs.indiana.edu:8045/GCWS/services/Echo"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Namespaces

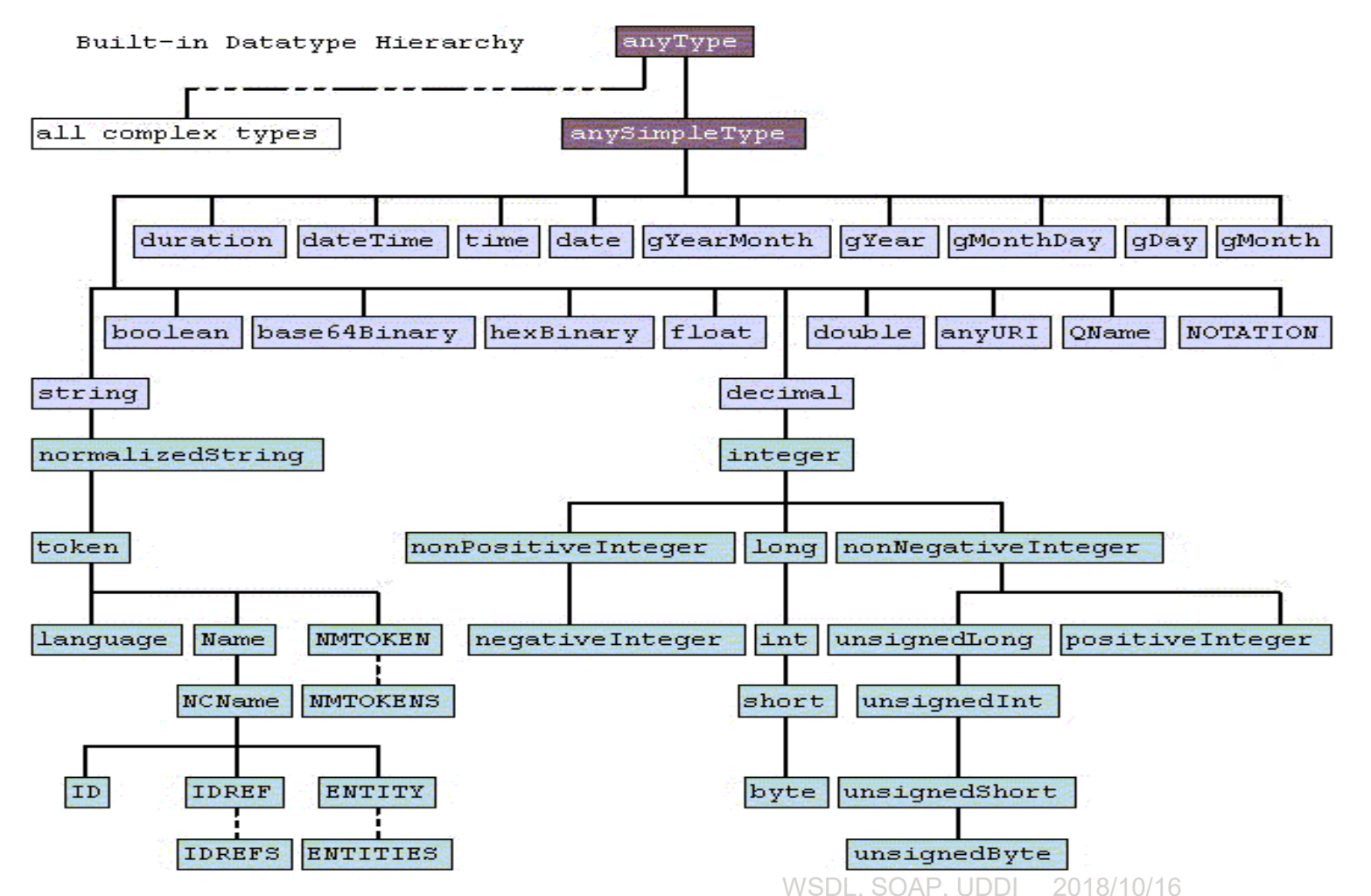
- The WSDL document begins with several XML namespace definitions.
- Namespaces allow you to compose a single XML document from several XML schemas.
- Namespaces allow you to identify which schema an XML tag comes from.
  - Avoids name conflicts.

WSDL TYPES

WSDL Types

- EchoService just has string messages.
  - So no special types definitions are needed in our WSDL.
- Strings are an XML schema built-in type.
  - See earlier XML lectures.

Schema Built In Types



When Would I Need a Type?

- Any time your Web Service needs to send data formatted by anything other than XML built-in types, you must define the type in WSDL.
- Example: Arrays are not built-in types!
  - Arrays of strings, ints, etc., must be defined in the WSDL <type></type> structure.

How Does WSDL Encode String Arrays?

- Imagine that my echo service actually echoes back an array of strings.
- Luckily for us, SOAP defines arrays, so we can import this definition.
- Next slide shows what this looks like.

String Array Example

```
<wsdl:types>
    <schema
        targetNamespace="http://grids.ucs.indiana.edu:8045/GCWS/services/EchoArray"
        xmlns="http://www.w3.org/2001/XMLSchema">
      <import                    namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="ArrayOf_xsd_string">
        <complexContent>
          <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType"
        wsdl:arrayType="xsd:string[]" />
          </restriction>
        </complexContent>
      </complexType>
      <element name="ArrayOf_xsd_string" nillable="true"  type="impl:ArrayOf_xsd_string" />
    </schema>
 </wsdl:types>
```

WSDL String Array Types

- WSDL <type> is nothing more than an extensibility placeholder in WSDL.
- Technically, the WSDL schema specifies that <type> </type> can contain a <sequence> of o or more <any> tags.
- And note that the <any> tag acts like wildcard.
  - You can insert any sort of xml here.

Inserting a Type

- Between <type></type>, we insert an <schema>.
- Since arrays are defined in SOAP encoding rules, I next *import* the appropriate schema.
  - "Import" means replaces a simple cut-and-past of the entire soap encoding schema.
- Next, insert our own local definition of a type called "ArrayOf_xsd_string".
- This is a restricted extension of the SOAP Array complex type.
  - We only allow 1 dimensional string arrays
  - It is also nillable—I am allowed insert a "null" value for the string.

Handling Other XML Types

- You can also express other message arguments as XML.
  - Examples: a purchase order, an SVG description of an image, a GML description of a map.
- In practice, these are handled by automatic Bean serializers/deserializers.
  - Castor is an example: http://www.castor.org/
  - XMLBeans is another http://xml.apache.org/xmlbeans/
- These are tools that make it easy to convert between XML and JavaBeans.
- By "JavaBeans" I mean objects that associate simple get/set methods with all data.
- Implementation dependent.

WSDL MESSAGES

Our Echo Messages

```
<wsdl:message   name="echoResponse">
   <wsdl:part name="echoReturn"
        type="xsd:string" />
</wsdl:message>
<wsdl:message   name="echoRequest">
   <wsdl:part name="ino"     type="xsd:string"
   />
</wsdl:message>
```

Remember the Echo Service?

- Our echo service takes a string argument and returns a string answer.
- In WSDL, I first abstract these as *messages.*
  - Echo needs two messages.
- Note we have not yet said message is the request and which is the response.
  - That is the job of the portType operations, coming up.

Structure of a Message

- WSDL <message> elements have name attributes and one or more *parts*.
  - The message name should be unique for the document.
  - <operation> elements will refer to messages by name.
- I need one <part> for each piece of data I need to send in that message.
- Each <part> is given a name and specifies its type.
  - <part> types can point to <wsdl:type> definitions if necessary.
  - Our service just needs xsd:strings, so no problem.

PORTTYPES AND OPERATIONS

EchoService portType

```
<wsdl:portType name="Echo">
   <wsdl:operation name="echo"
        parameterOrder="ino">
        <wsdl:input
        message="impl:echoRequest"
        name="echoRequest" />
        <wsdl:output
                message="impl:echoResponse"
        name="echoResponse" />
        </wsdl:operation>
 </wsdl:portType>
```

WSDL portTypes

- WSDL messages are only abstract messages.
  - We bind them to *operations* within the portType.
- The structure of the portType specifies (still abstractly) how the messages are to be used.

portType Message Patterns

- PortTypes support four types of messaging:
  - One way: Client send a message to the service and doesn't want a response.
    - <input> only.
  - Request-Response: Client sends a message and waits for a response.
    - <input>, then <output>
  - Solicit-Response: Service sends a message to the client first, then the client responds.
    - <output>, then <input>
  - Notification: <output> only.
- These still are abstract. We must implement them using some message protocol.
  - HTTP units of transmission are request and response, so mapping Solicit-Response to HTTP will take some work.

portType for EchoService

- The echo service has one method, echo.
- It takes one string argument and returns one string.
- In WSDL, the portType is “Echo”, the operation is “echo”.
- The messages are organized into input and output.
  - Messages are placed here as appropriate.
  - That is, <input> takes the <echoRequest> message.

Parameter Order

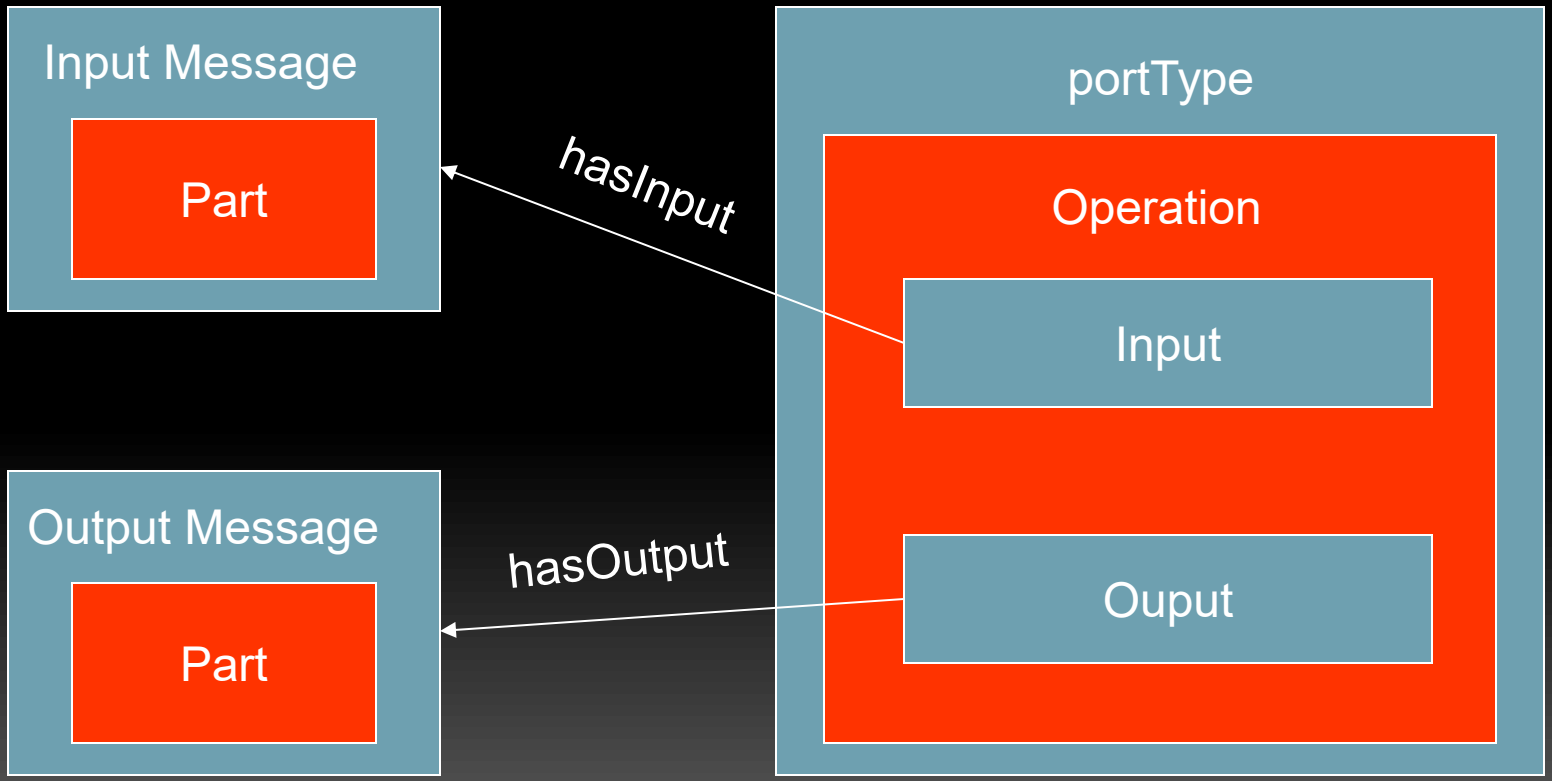
- This attribute of operation is used to specify zero or more space-separated values.
- The values give the order that the input messages must be sent.
- Echo is a bad example, since it only has one input parameter, named ***ino***.

WSDL Self-Referencing

- The WSDL <input> and <output> tags need to point back to the <message> definitions above:

```
<wsdl:message name="echoResponse">
    <wsdl:part name="echoReturn" type="xsd:string" />
 </wsdl:message>
...
<wsdl:portType name="Echo">
    <wsdl:operation name="echo" parameterOrder="ino">
            ...
            <wsdl:output message="impl:echoResponse"
            name="echoResponse" />
    </wsdl:operation>
 </wsdl:portType>
```

The Picture So Far…



BINDINGS

So Far…

- We have defined abstract messages, which have XML values.
  - Simple or custom-defined types.
- We have grouped messages into operations and operations into portTypes.
- We are now ready to bind the portTypes to specific protocols.

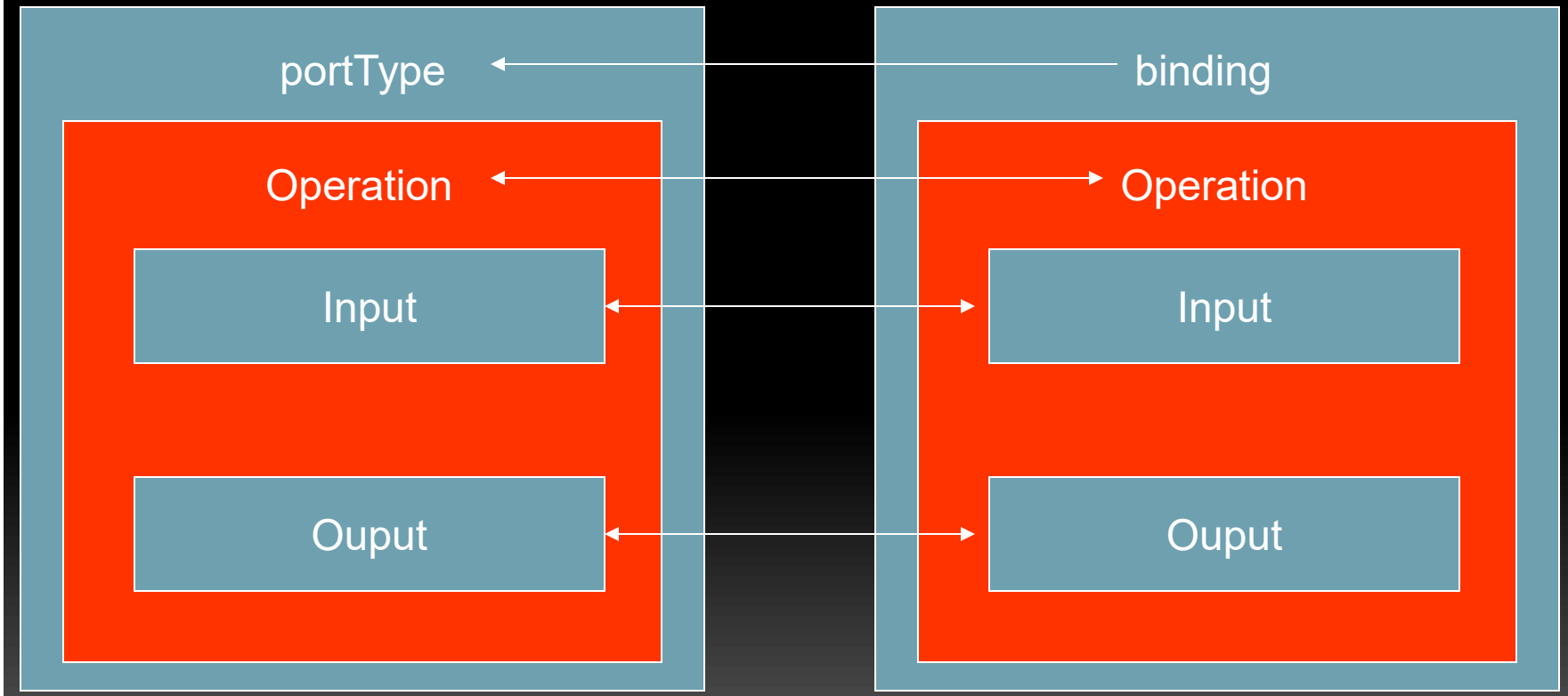
The Binding for Echo

```
<wsdl:binding name="EchoSoapBinding" type="impl:Echo">
 <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="echo">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="echoRequest">
      <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="..." use="encoded" />
    </wsdl:input>
    <wsdl:output name="echoResponse">
      <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="..." use="encoded" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

Binding tags

- Binding tags are meant to bind the parts of portTypes to sections of specific protocols.
  - SOAP, HTTP GET/POST, and MIME are provided in the WSDL specification.
- Bindings refer back to portTypes by name, just as operations point to messages.

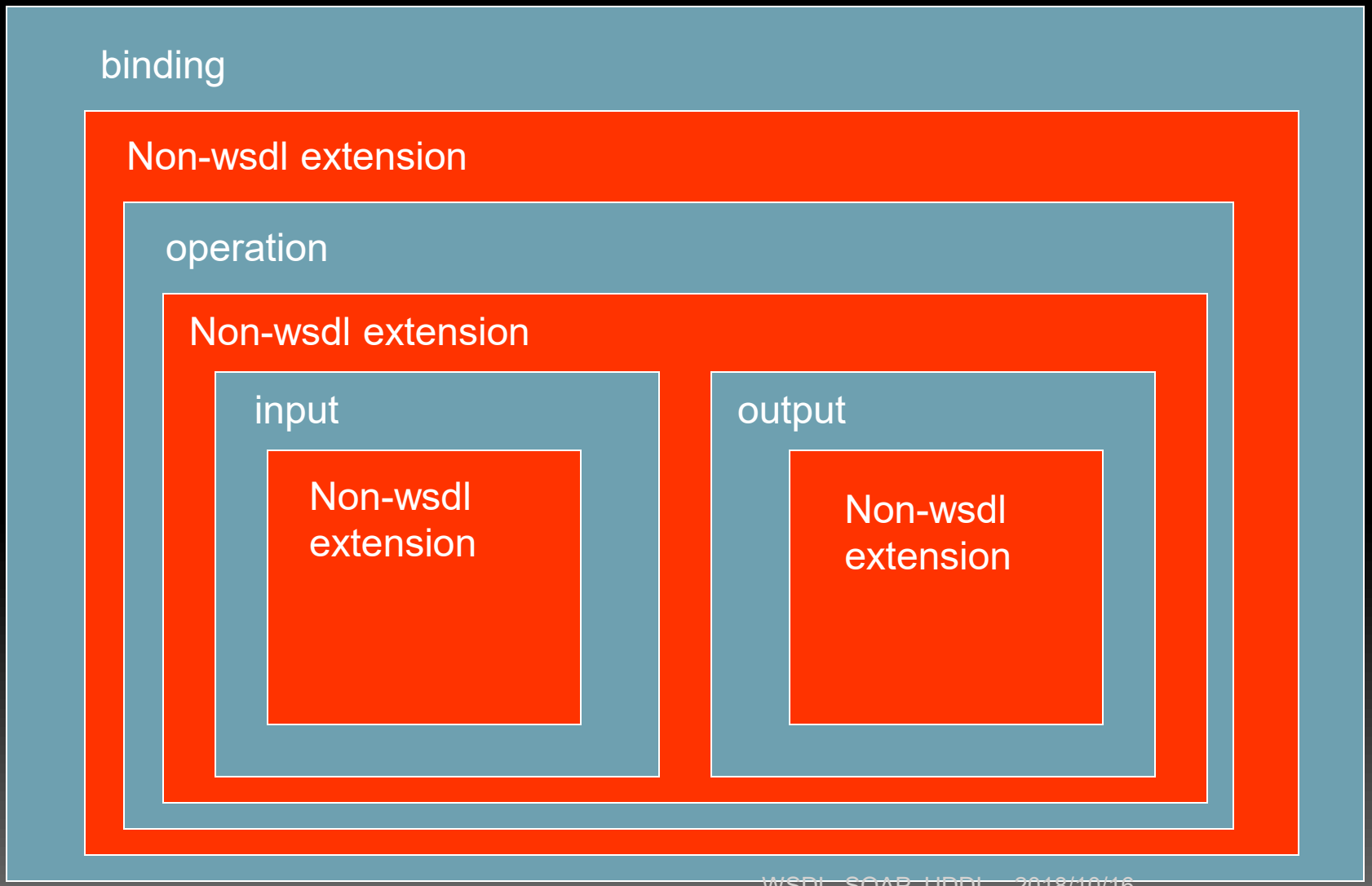
WSDL Internal References



Structure of the Binding

- <binding> tags are really just placeholders.
- They are meant to be extended at specific places by wsdl protocol bindings.
  - These protocol binding rules are defined in supplemental schemas.
- The following box figure summarizes these things
  - Green boxes are part of WSDL
    - From the wsdl namespace, that is.
  - Red boxes are parts of the document from other schemas
    - From wsdlsoap namespace in the echo example.

Binding Structure



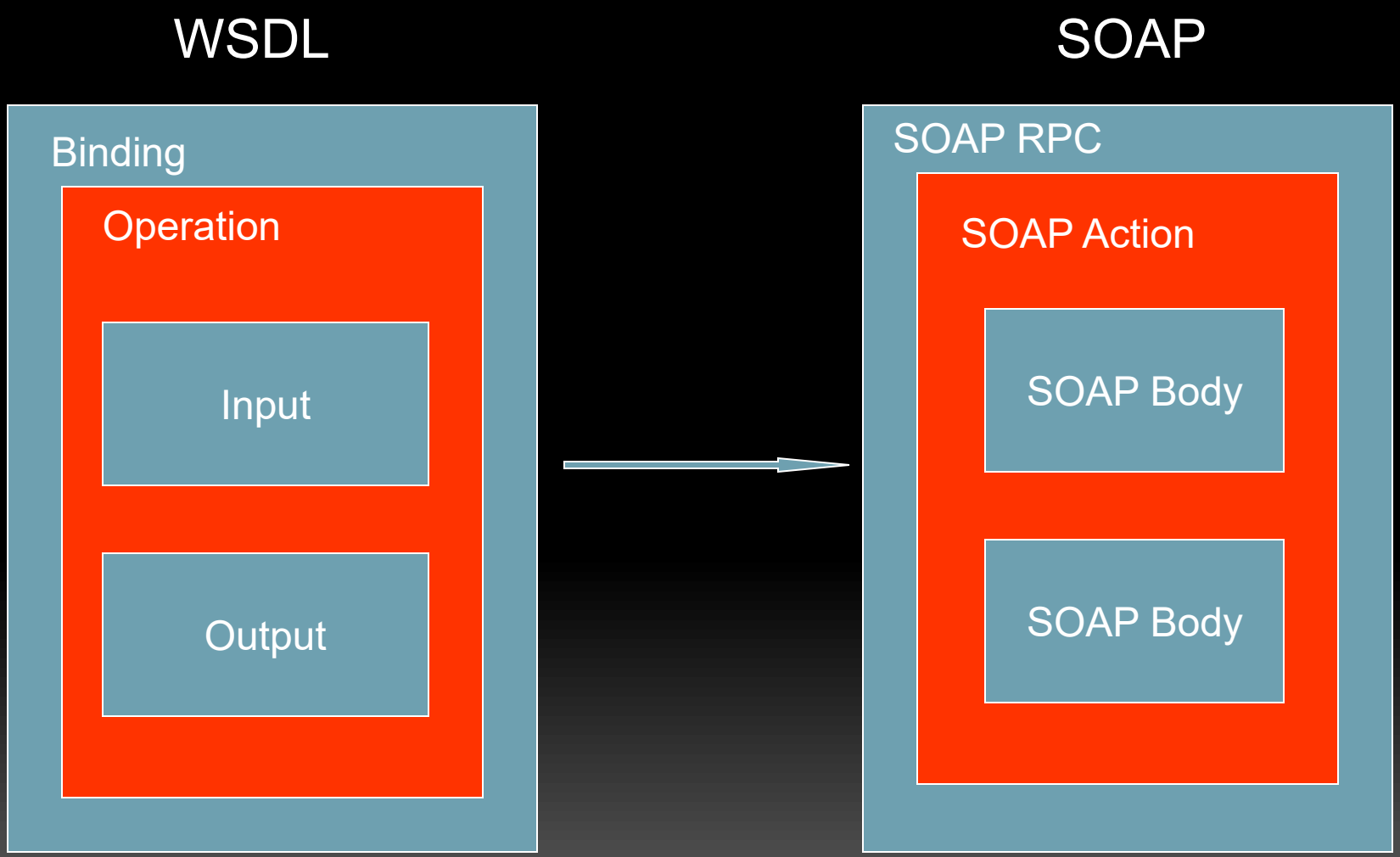
SOAP Bindings

- The WSDL bindings are meant to prescribe how the parts of the portType get transported.
- All the given bindings are to parts of SOAP messaging formats.
  - WSDL's SOAP bindings define mappings.
  - We will look at these in upcoming lectures.

For now, note the following

- We specify SOAP encoding
- SOAP is a message format and needs a transport protocol, so we specify HTTP.
- Operation styles may be either "RPC" or "Document".
  - We use RPC.
- SOAP Body elements will be used to actually convey message payloads.
  - RPC requires "encoded" payloads.
    - Each value (echo strings) is wrapped in an element named after the operation.
    - Useful RPC processing on the server side.
  - Documents are literal (unencoded)
    - Use to just send a payload of XML inside SOAP.

Binding Associations to SOAP



Binding Restrictions

- Binding elements point by name to portTypes.
- WSDL allows more than one binding element to point to the same port type.
  - Why?
  - Because a service may support multiple, alternative protocol bindings.

What Does It Mean?

- WSDL is not a programming language.
- A service that exposes an WSDL interface is just telling a client what it needs to do to communicate with the service.
  - Send me strings and I will return strings.
  - I expect SOAP messages that include the strings in the body.
  - I expect this body to be RPC encoded with the operation name so that I will know which operation the body contents belong to.
  - I will return SOAP messages that include Strings in the body.
  - These will also be encoded so that you know what to do with them.

PORTS AND SERVICES

Ports and Services

```
<wsdl:service name="EchoService">
   <wsdl:port
      binding="impl:EchoSoapBinding"
      name="Echo">
     <wsdlsoap:address
      location="http://..../"/>
  </wsdl:port>
</wsdl:service>
```

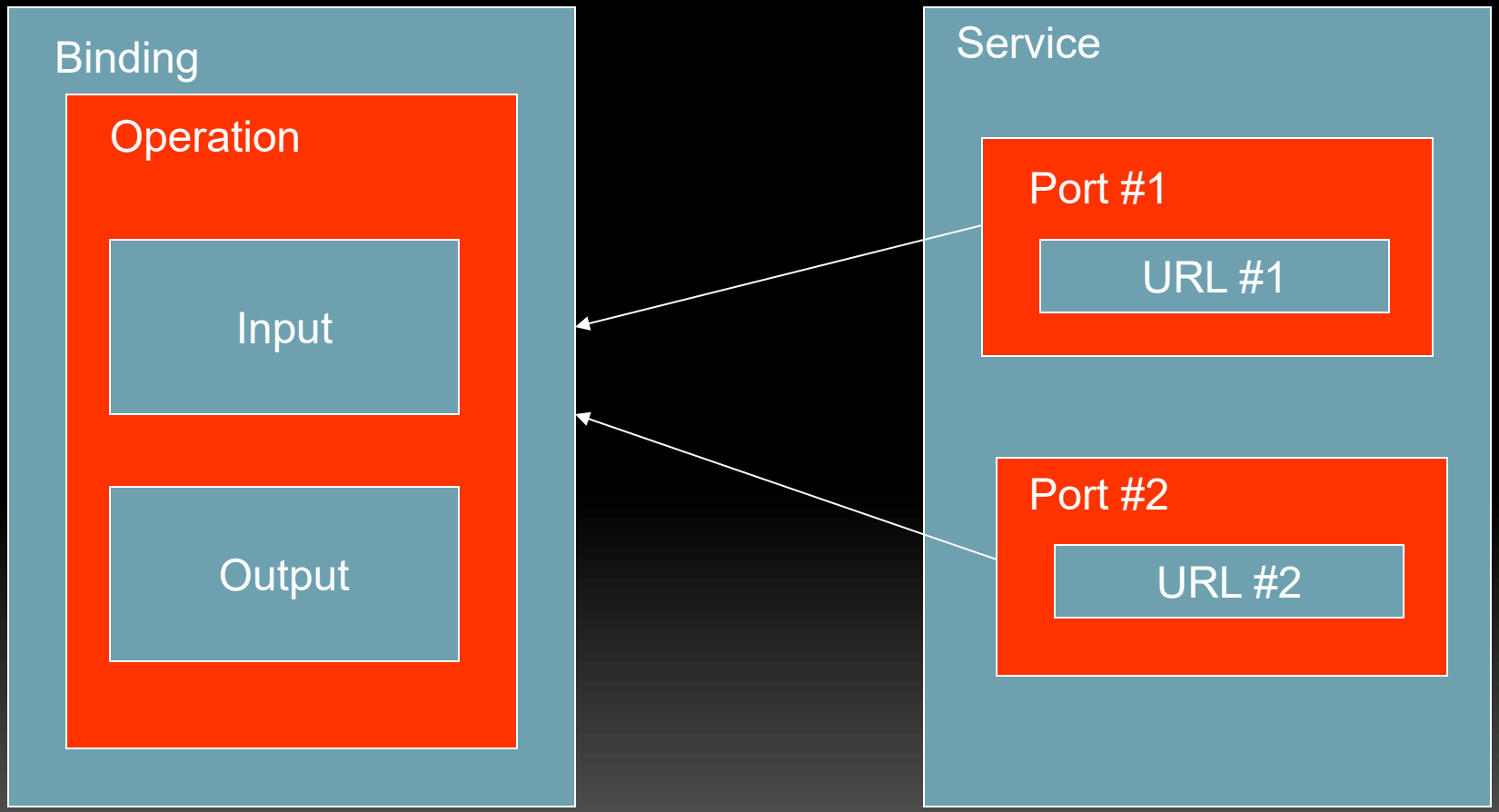
Port and Service Tags

- The service element is a collection of ports.
  - That's all it is for.
- Ports are intended to point to actual Web service locations
  - The location depends on the binding.
  - For SOAP bindings, this is a URL.

Ports and Services

- A service can have more than one port.
- Two ports can point back to the same binding element.
  - Ports refer to bindings by name
  - This allows you to provide alternative service locations.
- The figure on next slide conceptually depicts associating two ports to a single binding.
  - The ports differ only in the URLs of their services.

Port Associations to Bindings



SOAP

SIMPLE OBJECT ACCESS PROTOCOL

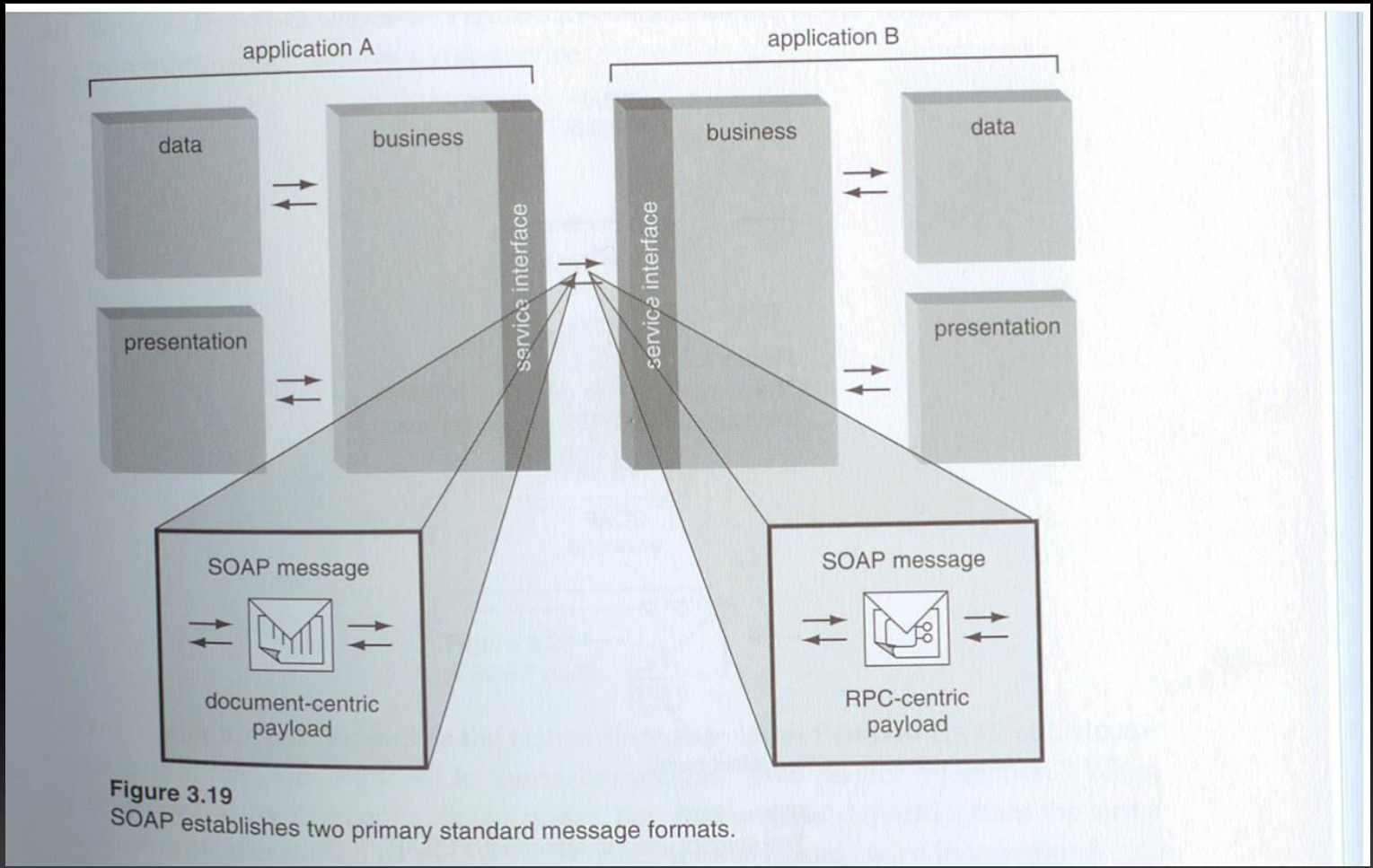


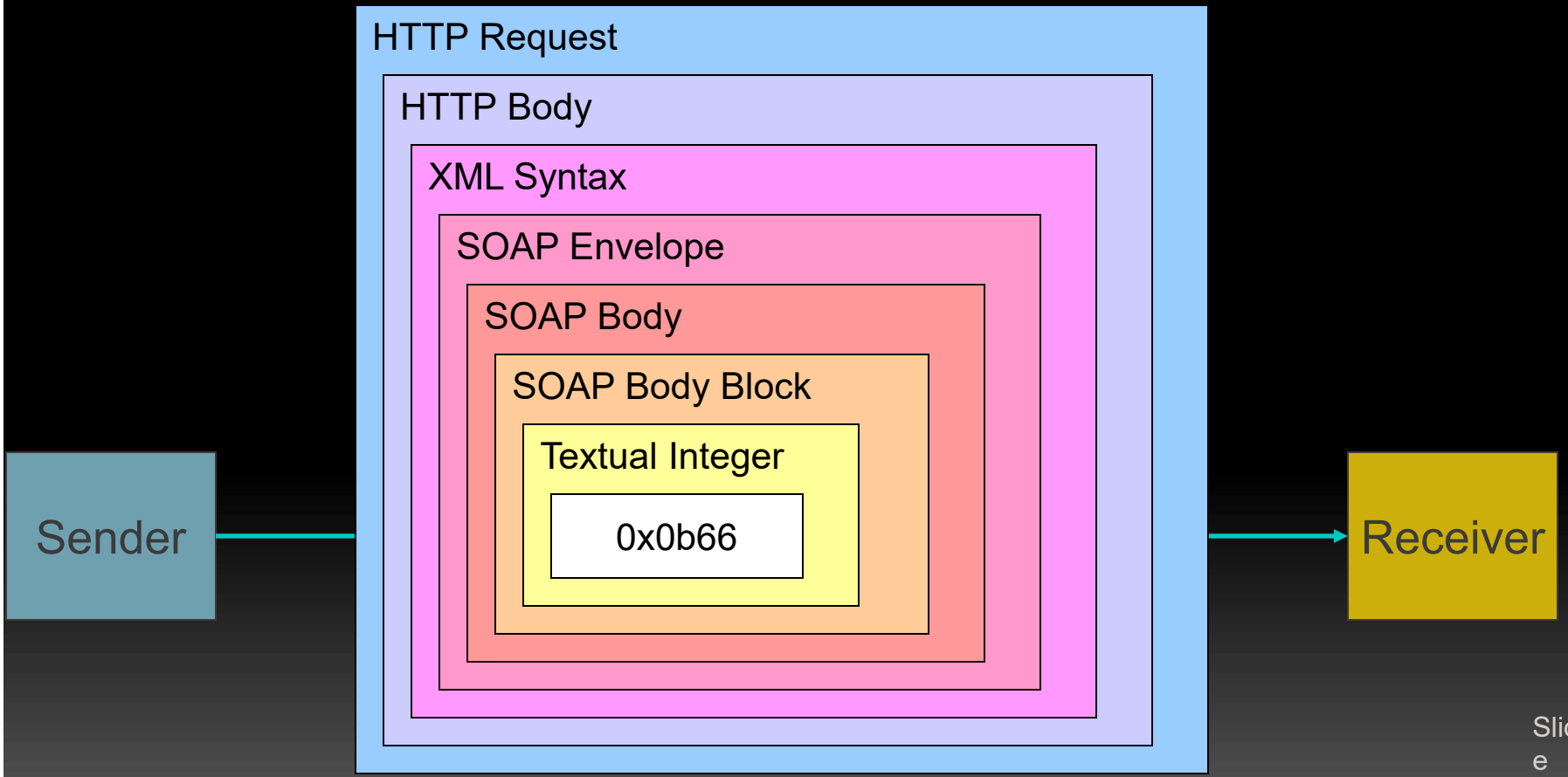
Figure 3.19
SOAP establishes two primary standard message formats.

Message Anatomy

Message Representation

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/soap-envelope">
  <env:Header>
    <data:headerBlock
      xmlns:data="http://example.com/header"
      env:actor="http://example.com/actor"
      env:mustUnderstand="true">
      ...
    </data:headerBlock>
    ...
  </env:Header>
  <env:Body>
    <data:bodyBlock xmlns:data="http://example.com/header">
      ...
    </data:bodyBlock>
    ...
  </env:Body>
</env:Envelope>
```

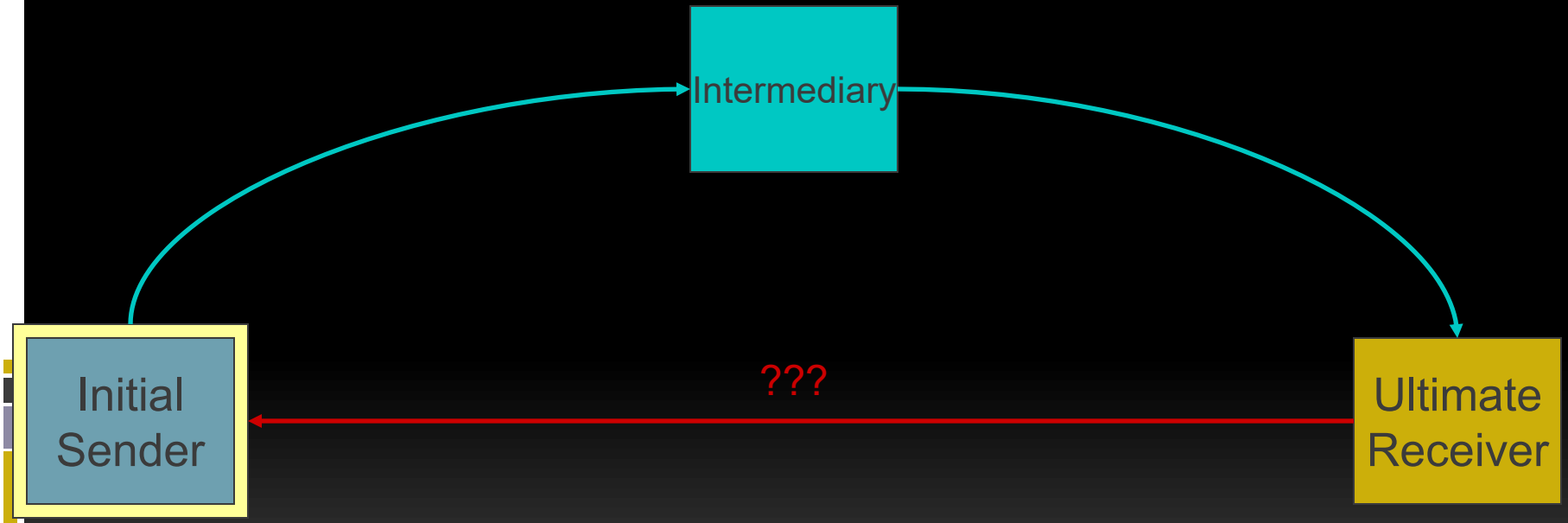
(In)Efficiency



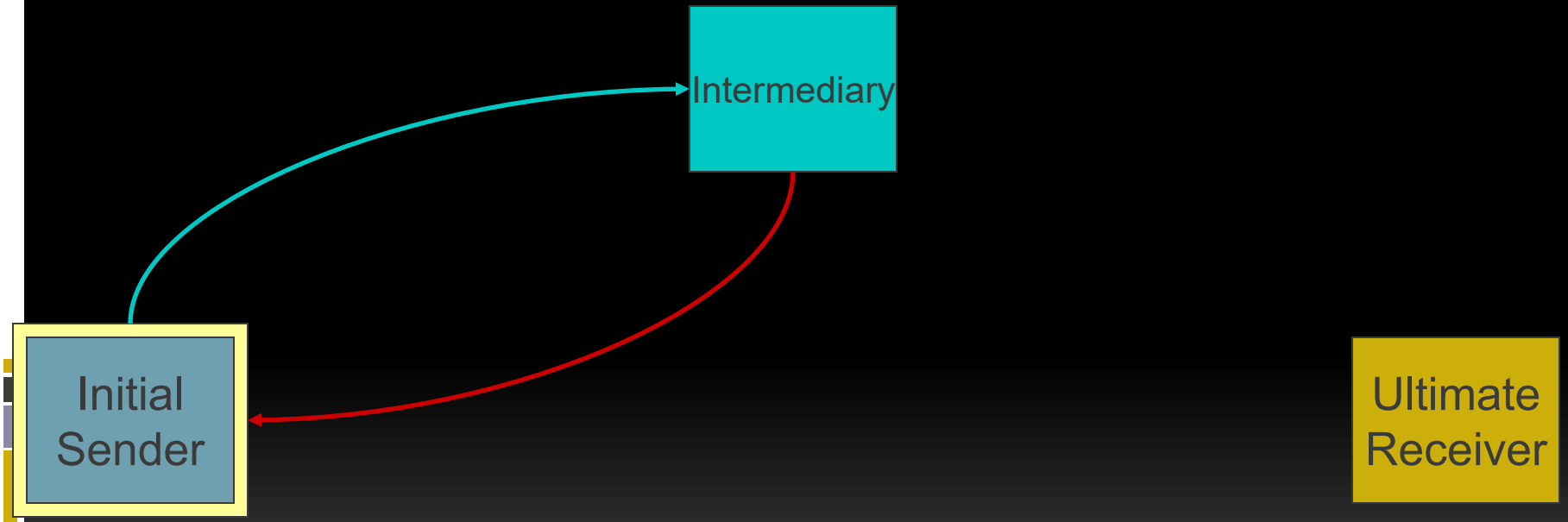
Exchange Paradigm

- Point-to-point exchange
  - Sender, receiver, possible intermediaries
- Uni-directional message exchange
  - True, but…
    - Specification includes semantics for dealing with faults
    - Faults cannot be ignored
    - Faults must be reported to the sending node

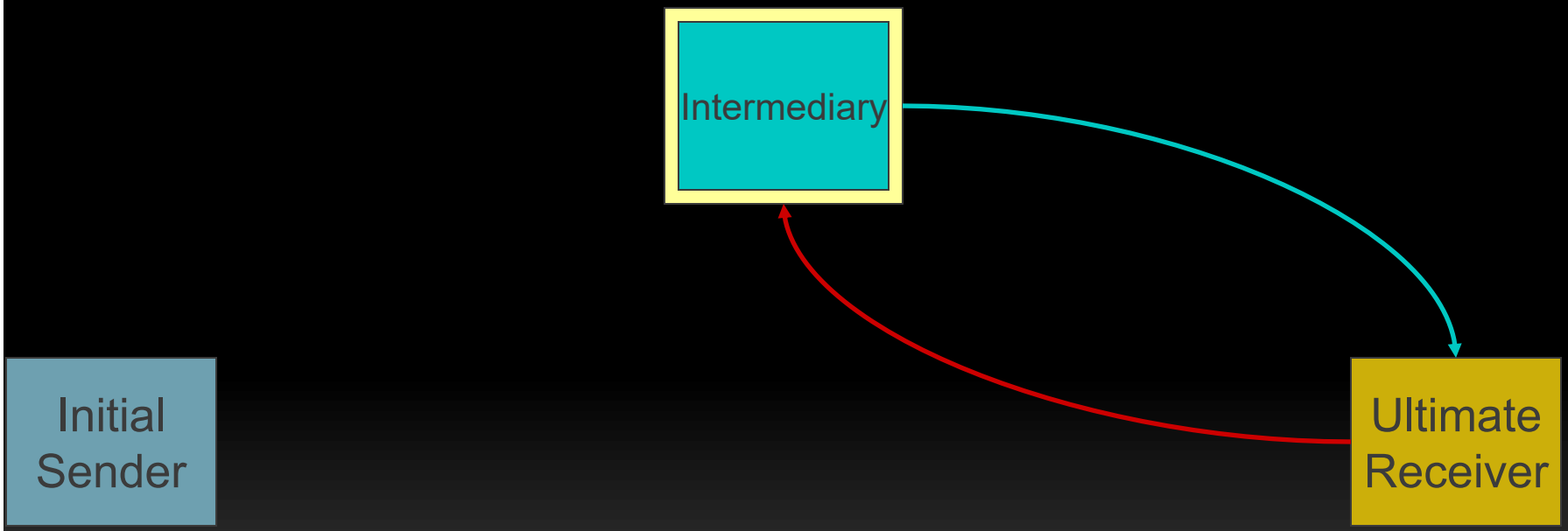
Uni-directional Exchange



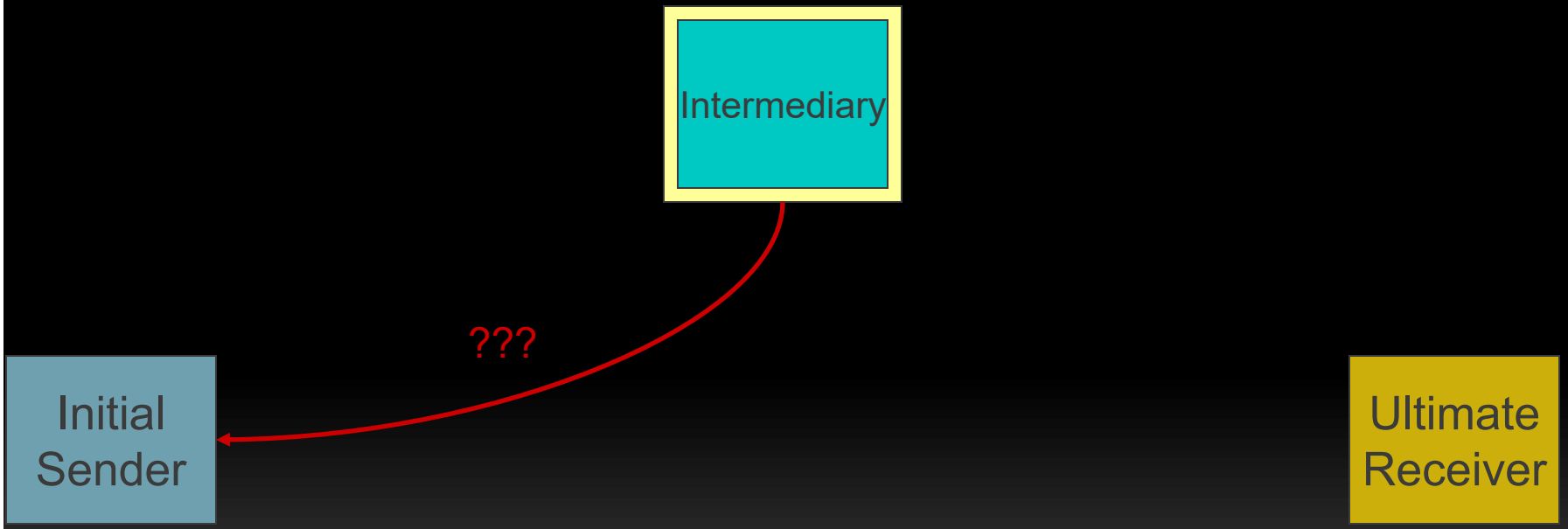
Bi-directional Exchange (Series)



Bi-directional Exchange (Series)



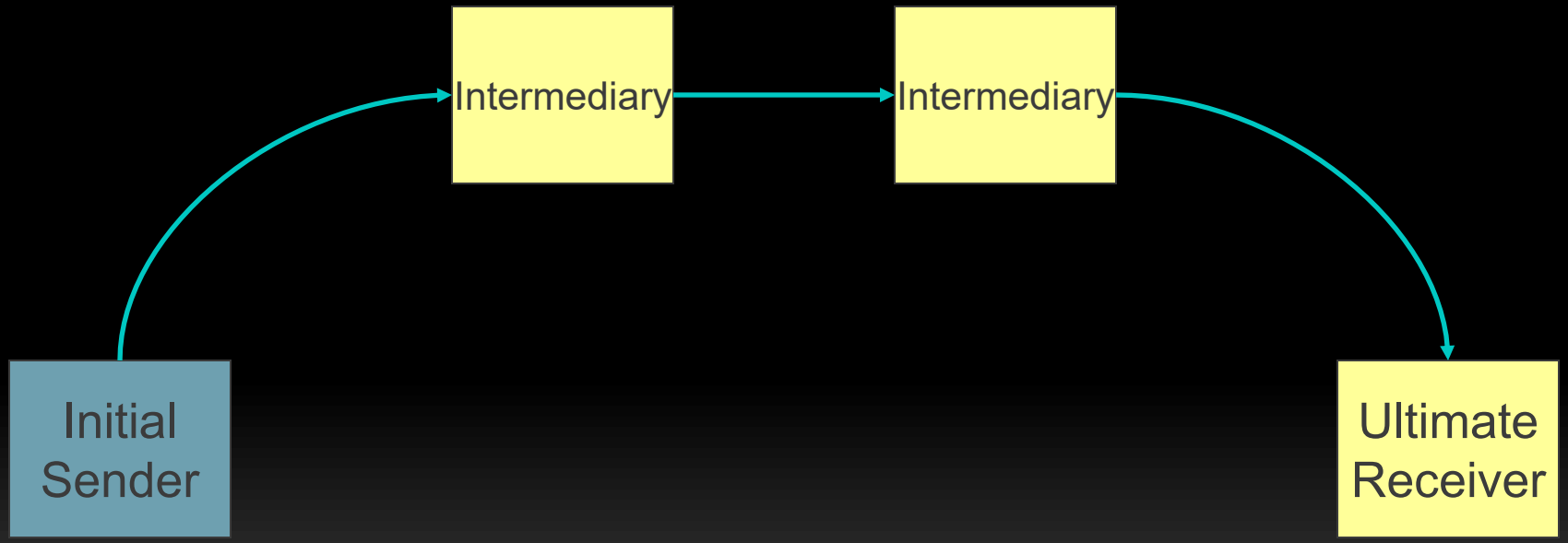
Bi-directional Exchange (Series)



Processing Model

- Point-to-point (sender-to-receiver) exchange, possibly via intermediaries
- Receivers assume "roles" as actors
- Header blocks can be specific to actors
  - Body blocks are always specific to the ultimate receiver
- Actors can be required to understand header blocks

Nodes


Intermediary
Intermediary
Initial Sender
Ultimate Receiver

Actor-specific Header Blocks

```
<data:headerBlock
  xmlns:data="http://example.com/header"
  env:actor="http://example.com/actor1"
  env:mustUnderstand="true">
  ...
</data:headerBlock>
<data:headerBlock
  xmlns:data="http://example.com/header"
  env:actor="http://example.com/actor2"
  env:mustUnderstand="true">
  ...
</data:headerBlock>
<data:headerBlock
  xmlns:data="http://example.com/header"
  env:actor="http://example.com/actor2"
  ...
</data:headerBlock>
```

Intermediary Algorithm

- Receive message
- Process appropriate header blocks
  - Processing possibly produces a fault
- Remove processed headers
- Add new headers
- Send message

Ultimate Recipient Algorithm

- Receive message
- Process appropriate header blocks
  - Processing possibly produces a fault
- Process all body blocks
  - Processing possibly produces a fault

Higher-level Exchange Paradigms

- RPC
  - Fits well with HTTP 1.1 binding
  - Current activity within the XML Protocol Working Group
- Conversational
  - Fits well with general message passing, but awkward with HTTP 1.1 binding

Normative References

- http://www.w3.org/2000/09/XML-Protocol-Charter
- http://www.w3.org/2002/ws/Activity.html
- http://www.w3.org/TR/xmlp-reqs/
- http://www.w3.org/TR/xmlp-am/
- http://www.w3.org/TR/xmlp-scenarios/
- http://www.w3.org/TR/soap12-part0/
- http://www.w3.org/TR/soap12-part1/
- http://www.w3.org/TR/soap12-part2/

UDDI

UNIVERSAL DESCRIPTION DISCOVERY & INTEGRATION

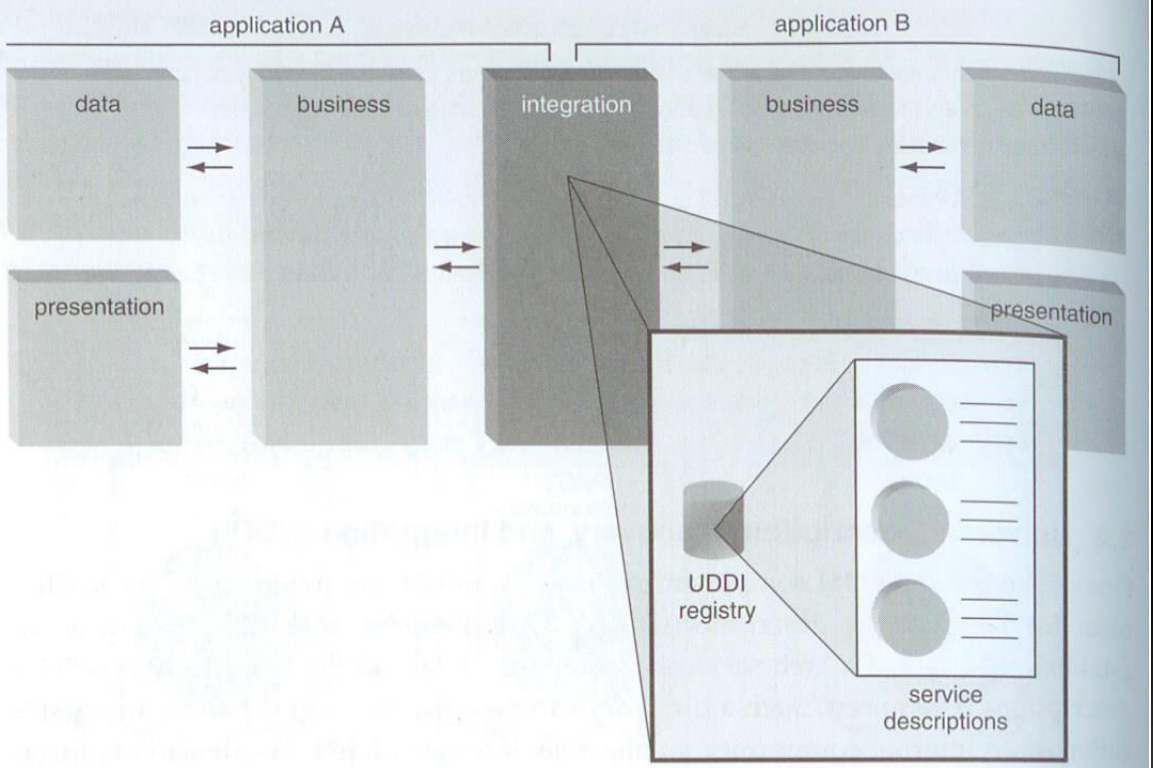


Figure 3.28
Service descriptions centralized in a private UDDI registry

A Registry for WSs: UDDI

- **Universal Description Discovery and Integration (UDDI)**
  - **Unified and systematic way to find service providers**
  - **roughly equivalent to "phone directory" of web services**
- **Specifications**
  - **Schemas for service and business description**
  - **Query and update API for the registry**
- **WS-I compatible**
- **Based on XML, HTTP, IP, SOAP, WSDL standards**
- **Current status :**
  - UDDI 3.0 has been released in August 2003.
  - OASIS UDDI Specifications Technical Committee manages and develops UDDI Specifications

UDDI Registries Organizing Structure

- UDDI registry entries store published information about WSs.
  - White Pages: information such as name, address, i.e. contact details
  - Yellow Pages: information such as categorization of businesses or services
  - Green Pages: information such as technical data about services

UDDI defines entities to describe businesses and their services - I

- businessEntity
  - provides information, including identifiers, contact information etc…

    [white-pages information]
  - includes one or more businessService (service entity) elements that represents the services it provides
  - specifies a categoryBag to categorize the business [yellow-pages information]
  - a unique key identifies each businessEntity

A simple businessEntity structure

Business Key

Business Name

```
<businessEntity businessKey=
        “A687FG00-56NM-EFT1-3456-098765432124”>
  <name>Acme Travel Incorporated</name>
  <description xml:lang=“en”>
        Acme is a world leader in online travel services
  </description>
  <contacts>
        <contact useType=“US general”>
        <personName>Acme Inc.</personName>
        <phone>1 800 CALL ACME</phone>
        <email useType=“”>acme@acme-travel.com</email>
        <address>….</address>
        </contact>
  </contacts>
```

Services

```
  <businessServices>
    ...
    ….
  </businessServices>
```

Category

```
  <identifierBag> ...
  </identifierBag>
  <categoryBag> ...
  <keyedReference tModelKey=
        “UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384”
        keyName=“Electronic check-in”
        keyValue=“84121801”/>
  </categoryBag>
</businessEntity>
```

UDDI defines entities to describe businesses and their services - II

- businessService (service entity)
  - includes information such as name, description. [white-pages information]
  - uniquely identified by a service key
  - specifies a categoryBag to categorize the service [yellow-pages information]
  - contains a list of bindingTemplates which in turn contains tModelInstanceDetails encoding the technical service information [green-pages information]
  - includes reference to its host with a businessKey

A simple businessService structure

Service Key

```
<businessService serviceKey=
        "894B5100-3AAF-11D5-80DC-002035229C64"
        businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
  <name>ElectronicTravelService</name>
  <description xml:lang="en">Electronic Travel Service</description>
  <bindingTemplates>
        <bindingTemplate bindingKey=
                "6D665B10-3AAF-11D5-80DC-002035229C64"
                serviceKey="89470B40-3AAF-11D5-80DC-002035229C64">
          <description>
                SOAP-based e-checkin and flight info
          </description>
          <accesssPoint URLType="http">
                http://www.acme-travel.com/travelservice
          </accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo tModelKey="D2033110-3BGF-1KJH-234C-09873909802">
             ...
            </tModelInstanceInfo>
          </tModelInstanceDetails>
        </bindingTemplate>
  </bindingTemplates>
  <categoryBag>
    ...
  </categoryBag>
</businessService>
```

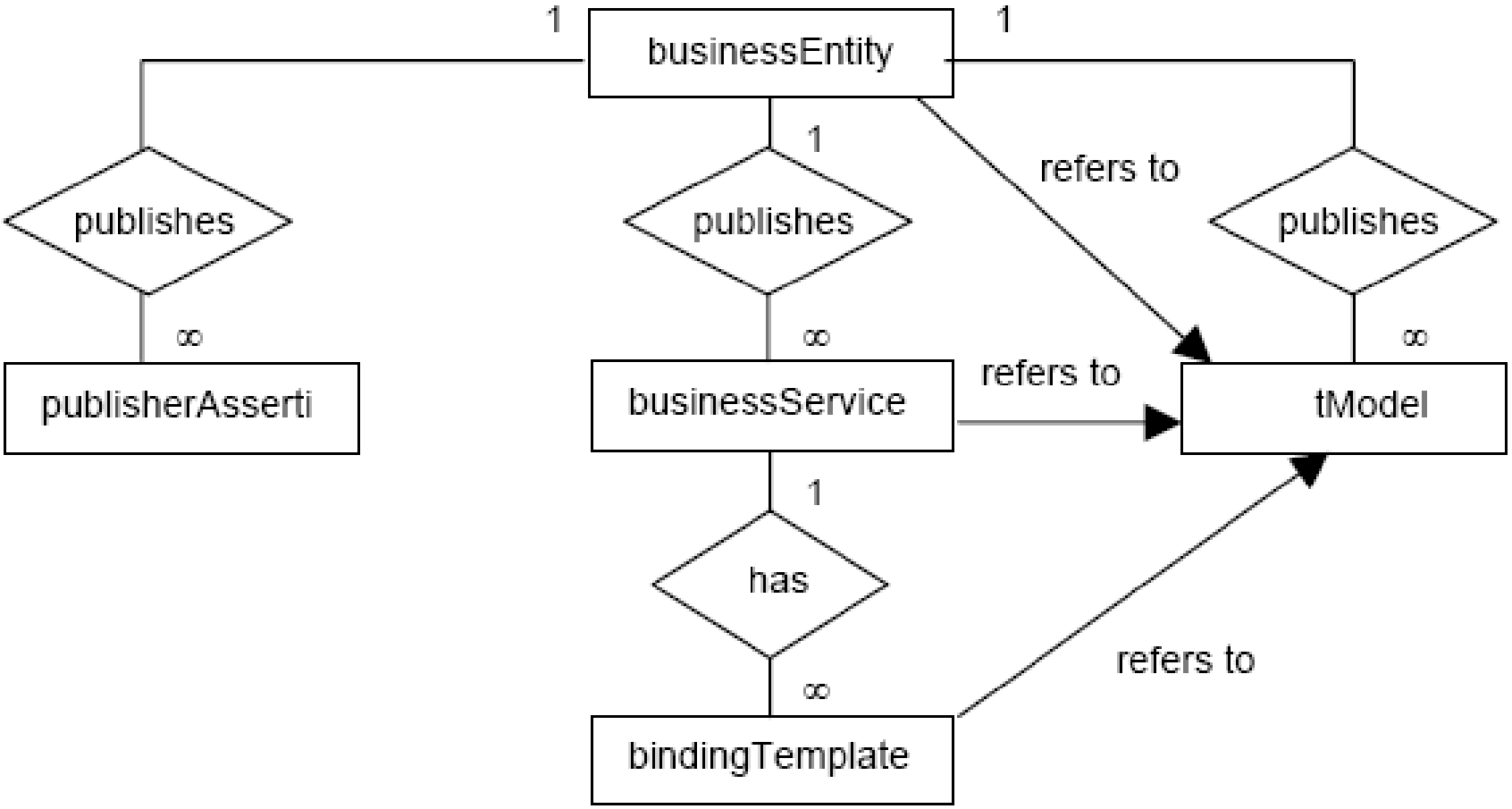
Service Name

Binding Template

tModelDetails

Category

UDDI Information Model

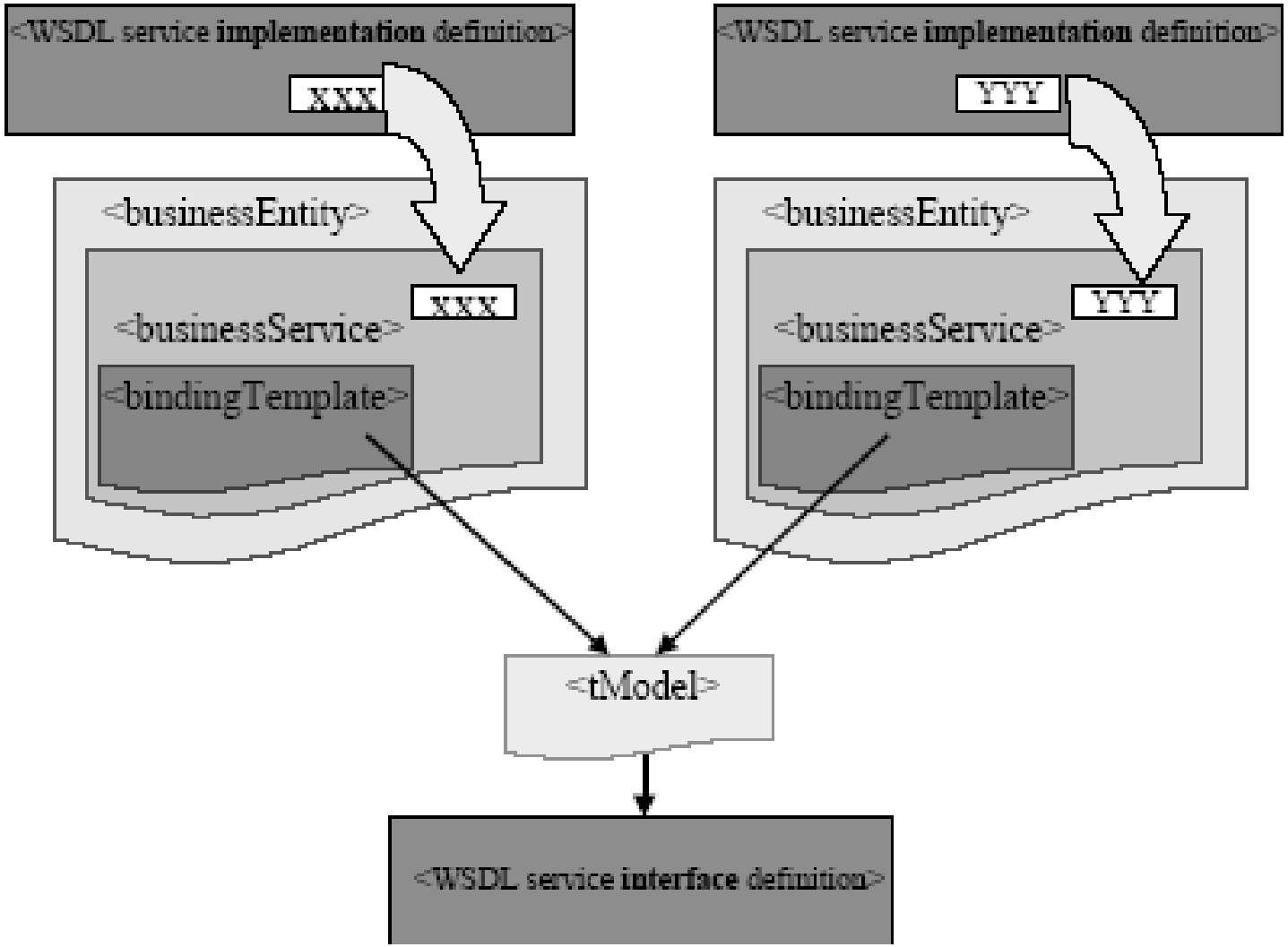


Information model of UDDI

UDDI Query

- UDDI Search API allows users to query for service providers that provide particular service.
- A UDDI query may be for
  - A business entity
    - Using business name, business key or business category (i.e. find_business())
  - A list of publisher assertions
    - Using business key (i.e. find_relatedBusiness())
  - A business service
    - Using the business key of service key and service name (i.e. find_service())
  - Service key of a bussiness entity
    - Using a binding template (i.e. find_binding())
  - A set of business entities and business services adopting same tModel
    - Using a tModel (i.e. find_tModel())
- After finding the requited UDDI entry, a set of API is used to get details of those entries from UDDI
  - get_businessDetail(), get_serviceDetail(), get_bindingDetail(), get_tModelDetail()

UDDI and WSDL relationship



UDDI and WSDL Relationship.

tModel

- A WSDL document is registered as a tModel into UDDI registry
  - In order to describe a service in more expressive way, an external information is referenced where the type and format of this information should be arbitrary.
  - UDDI Specs. leaves the responsibility of defining such arbitrary information types and formats to programmers.
- tModel is a UDDI construct to refer an interface describing WSDL document.
- The tModel idea:
  - to better describe a service we tend to reference information
  - such information type or format should not be anticipated
  - replacing such information about a service with a unique key provides a reference to arbitrary information types

tModels for Categorization

- Using categorization, UDDI directory can be queried for specific type of services.
- Each classification in a taxonomical system is registered as a tModel.
- Three standard taxonomies cited by UDDI are
  - North American Industry Classification System (NAICS) taxonomy – an industry classification
  - The Universal Standard Products and Services Code System (UNSPSC) taxonomy – a classification of products and services
  - The International Organization for Standardization Geographic taxonomy (ISO 3166)

A simplified tModel definition

```
<tModel tModelKey=“”>
   <name>http://www.travel.org/e-checkin-interface</name>
   <description xml:lang=“en”>
          Standard service interface definition for travel services
   </description>
   <overviewDoc>
          <description xml:lang=“en”>
                 WSDL Service Interface Document
          </description>
          <overviewURL>
                 http://www.travel.org/services/e-checkin.wsdl
          </overviewURL>
   </overviewDoc>
   <categoryBag> ...
   </categoryBag>
</tModel>
```

An example on How do we bind a WSDL to UDDI? Step by Step

```
ml version="1.0" encoding="utf-8"?>
finitions name="StockQuote" targetNamespace="http://example.com/stockquote/" xmlns:tns=http://example.com/stockquote/
lns:xsd1="http://example.com/stockquote/schema/" xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
mlns="http://schemas.xmlsoap.org/wsdl/">
<types>
 <schema
    targetNamespace="http://example.com/stockquote/schema/"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="TradePriceRequest">
      <complexType><all><element name="tickerSymbol" type="string"/></all></complexType>
    </element>
    <element name="TradePrice">
      <complexType><all><element name="price" type="float"/></all></complexType>
    </element>
 </schema>
/types>
message name="GetLastTradePriceInput">
 <part name="body" element="xsd1:TradePriceRequest"/>
/message>
message name="GetLastTradePriceOutput">
 <part name="body" element="xsd1:TradePrice"/>
/message>
portType name="StockQuotePortType">
 <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
 </operation>
/portType>
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
 ....
/binding>
<service name="StockQuoteService"> <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
    <soap:address location="http://location/sample"/> </port>
/service>
efinitions>
```

"Input\Output" Details

input
output

A sample WSDL file

- 1 portType
- 1 binding
- 1 service
- 1 port

"Method" Specification

"Binding" Details

UDDI portType tModel

```
<tModel tModelKey="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3" >
   <name>
      StockQuotePortType
   </name>
   <overviewDoc>
      <overviewURL>
         http://location/sample.wsdl
      <overviewURL>
   <overviewDoc>
   <categoryBag>
      <keyedReference
         tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
      keyName=”portType namespace”
         keyValue=”http://example.com/stockquote/” />
      <keyedReference
         tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
      keyName=”WSDL type”
         keyValue="portType" />
   </categoryBag>
</tModel>
```

tModel name

overviewDoc

categoryBag

UDDI binding

tModel

```
odel tModelKey="uuid:49662926-f4a5-4ba5-b8d0-32ab388dadda">
name>
  StockQuoteSoapBinding
/name>
<overviewDoc>
  <overviewURL>
    http://location/sample.wsdl
  </overviewURL>
/overviewDoc>
categoryBag>
  <keyedReference
    tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
   keyName=”binding namespace”
    keyValue=”http://example.com/stockquote/” />
  <keyedReference
    tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
   keyName=”WSDL type”
    keyValue="binding" />
  <keyedReference
    tModelKey="uuid:082b0851-25d8-303c-b332-f24a6d53e38e"
   keyName=”portType reference”
    keyValue="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3" />
  <keyedReference
    tModelKey="uuid:4dc74177-7806-34d9-aecd-33c57dc3a865"
   keyName=”SOAP protocol”
    keyValue= “uuid:aa254698-93de-3870-8df3-a5c075d64a0e” />
  <keyedReference
    tModelKey="uuid:e5c43936-86e4-37bf-8196-1d04b35c0099"
   keyName=”HTTP transport”
    keyValue=" uuid:68DE9E80-AD09-469D-8A37-088422BFBC36" />
  <keyedReference
    tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
   keyName=”uddi-org:types”
    keyValue="wsdlSpec" />
/categoryBag>
Model>
```

overviewDoc

a keyedReference

categoryBag

UDDI businessService and bindingTemplate

```
<businessService
      serviceKey="102b114a-52e0-4af4-a292-02700da543d4"
      businessKey="1e65ea29-4e0f-4807-8098-d352d7b10368">
   <name>Stock Quote Service</name>
   <bindingTemplates>
      <bindingTemplate
            bindingKey="f793c521-0daf-434c-8700-0e32da232e74”
            serviceKey="102b114a-52e0-4af4-a292-02700da543d4">
         <accessPoint URLType="http">
            http://location/sample
         </accessPoint>
         <tModelInstanceDetails>
            <tModelInstanceInfo
               tModelKey="uuid:49662926-f4a5-4ba5-b8d0-32ab388dadda">
               <description xml:lang="en">
                  text here…
               </description>
               <instanceDetails>
                  <instanceParms>StockQuotePort</instanceParms>
               </instanceDetails>
            </tModelInstanceInfo>
            <tModelInstanceInfo
               tModelKey="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3">
               <description xml:lang="en">
                  The wsdl:portType that this wsdl:port implements.
               </description>
            </tModelInstanceInfo>
         </tModelInstanceDetails>
      </bindingTemplate>
   </bindingTemplates>
   <categoryBag>
      …..
   </categoryBag>
</businessService>
```

business name

bindingTemplate

tModelInstanceDetails

tModelInstanceInfo

Query example - I

- Find the businessService for a WSDL service

```
<find_service generic=”2.0” xmlns=”urn:uddi-org:api_v2”>
   <categoryBag>
      <keyedReference
            tModelKey=”uuid:6e090afa-33e5-36eb-81b7-1ca18373f457”
         keyName=”WSDL type”
            keyValue=”service” />
      <keyedReference
            tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
         keyName=”service namespace”
            keyValue=”http://example.com/stockquote/” />
      <keyedReference
            tModelKey="uuid:2ec65201-9109-3919-9bec-c9dbefcaccf6"
         keyName=”service local name”
            keyValue="StockQuoteService" />
   </categoryBag>
</find_service>
```

Query example - II

- Find tModel for portType name

```
<find_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
   <name>StockQuotePortType</name>
   <categoryBag>
      <keyedReference
       tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
       keyName="WSDL type"
       keyValue="portType"/>
      <keyedReference
       tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
       keyName="portType namespace"
       keyValue="http://example.com/stockquote/"/>
   </categoryBag>
</find_tModel>
```

Query example - III

- Find bindings for portType

```
nd_tModel generic=”2.0” xmlns=”urn:uddi-org:api_v2”>
<categoryBag>
   <keyedReference
    tModelKey=” uuid:6e090afa-33e5-36eb-81b7-1ca18373f457”
    keyName=”WSDL type”
    keyValue=”binding”/>
   <keyedReference
    tModelKey=”uuid:082b0851-25d8-303c-b332-f24a6d53e38e”
    keyName=”portType reference”
    keyValue=”uuid:e8cf1163-8234-4b35-865f-94a7322e40c3”/>
</categoryBag>
/find_tModel>
```

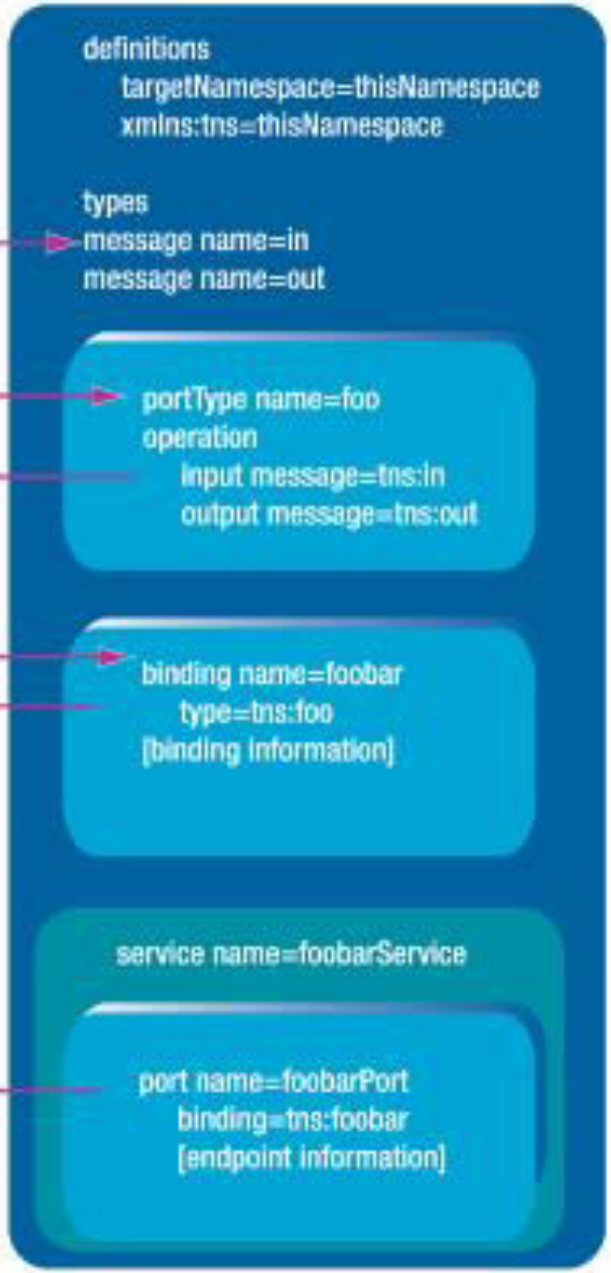
Recent Updates on UDDI Specs.

- richer mapping of WSDL 1.1 to UDDI v.2 and v.3
  - portType is supported.
    - Note that portType replaced by Interface in WSDL 2.
  - any logical/physical WSDL structure is supported

- new querying abilities on UDDI registries
  - Given the namespace of a wsdl:portType, find the tModel that represents that portType.
  - Given the namespace a wsdl:binding, find the tModel that represents that binding.
  - Given a tModel representing a portType, find all tModels representing bindings for that portType.
  - Given a tModel representing a portType, find all bindingTemplates that represent implementations of that portType.
  - Given a tModel representing a binding, find all bindingTemplates that represent implementations of that binding.
  - Given the namespace of a wsdl:service, find the businessService that represents that service.

http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v200-20031104.htm

WSDL data model



types contains data type definitions
messages consist of one or more parts

portType describes an abstract set of operations

binding describes a concrete set of
formats and protocols for the foo portType

port describes an implementation
of the foobar binding

FIGURE 2 | The WSDL data model

UDDI data model

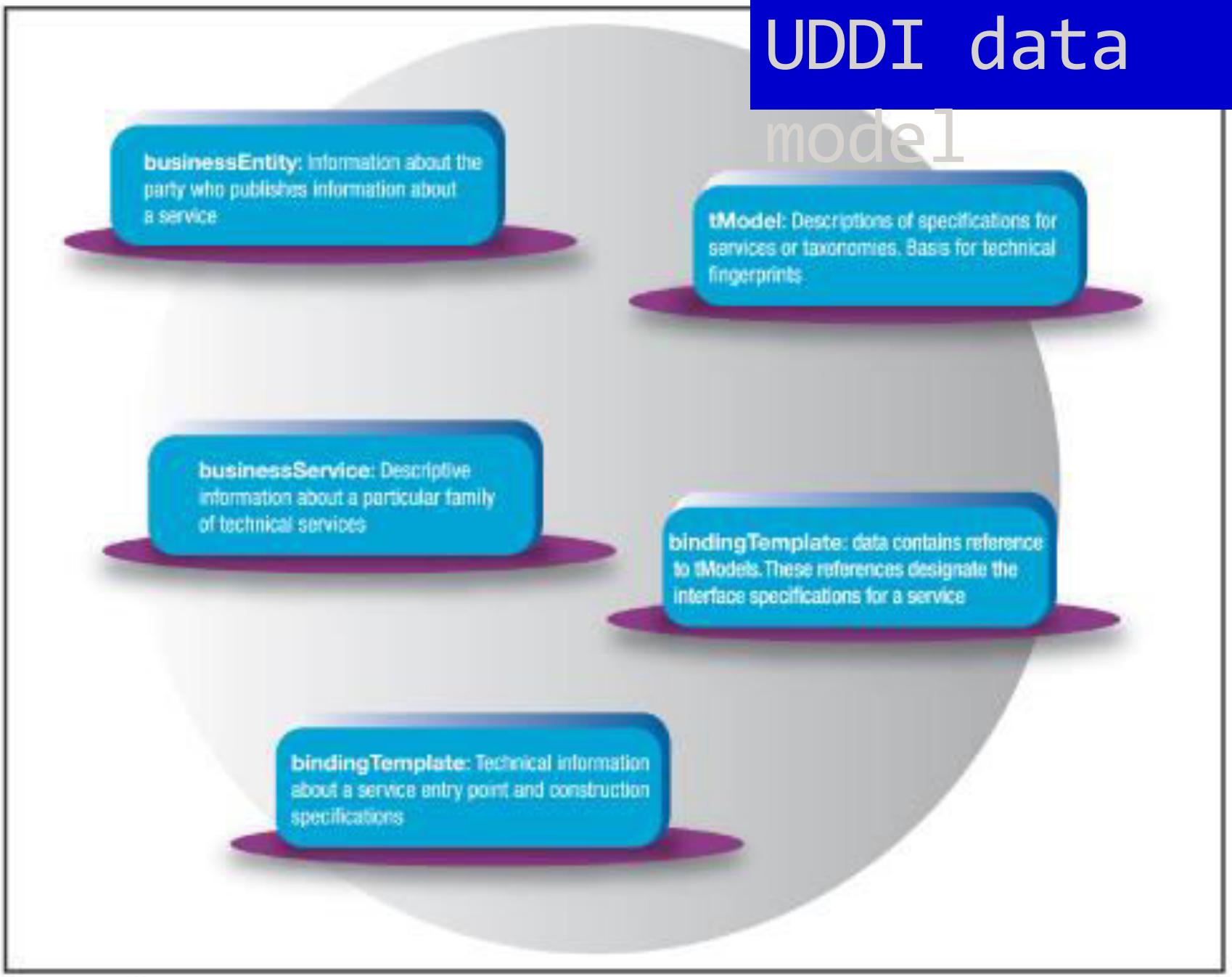
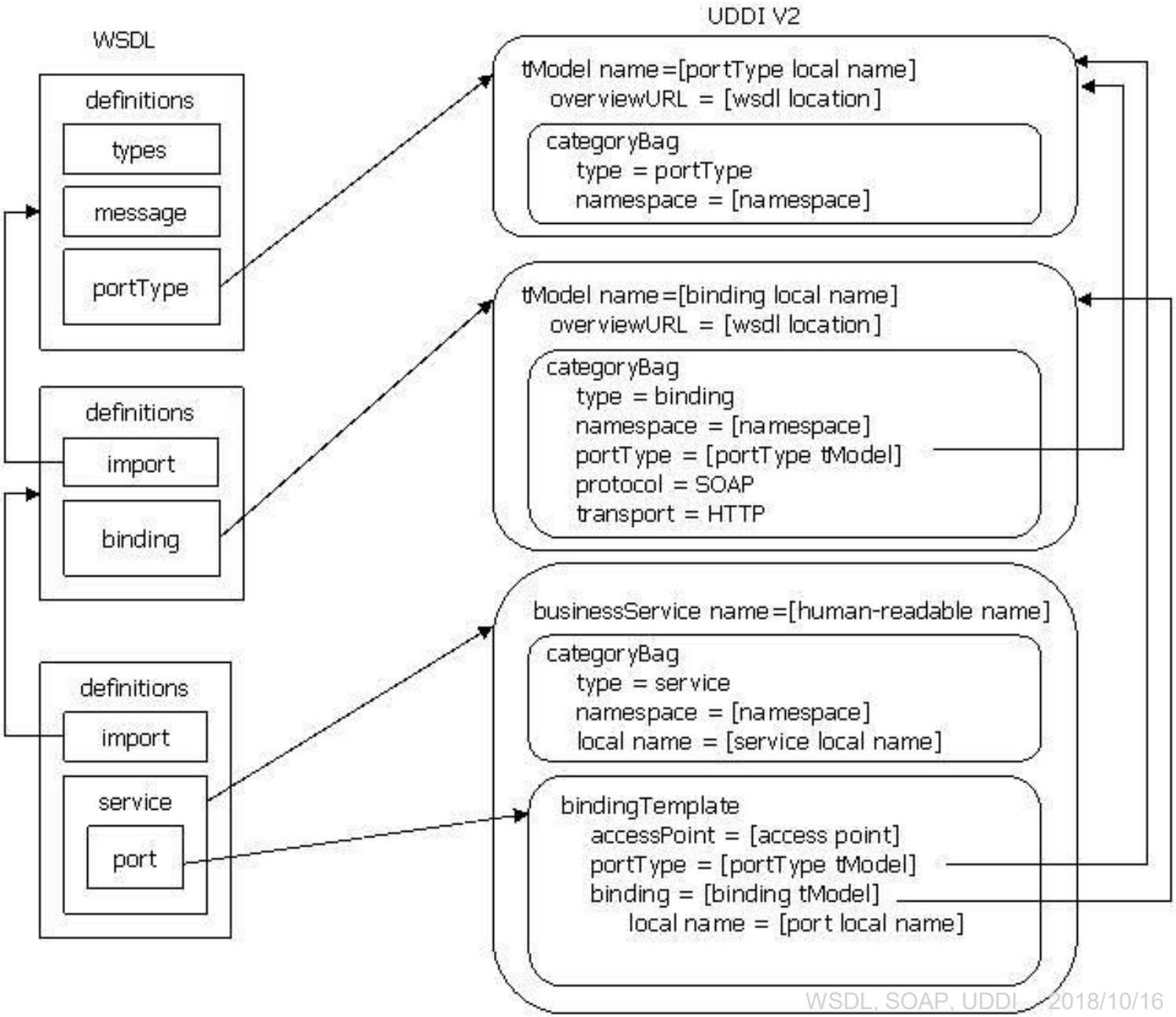


FIGURE 1 | The UDDI data model

mapping of WSDL 1.1 to UDDI V2 data model



Limitations of UDDI

- tModels are not stored in UDDI registries themselves. A unique identifier referencing a tModel is contained in the registries.
- There is no uniform way of querying about services, service interfaces and classifications.
- UDDI does not support WSDL security

More Limitations…

- Out-of-date service documents in UDDI registries. No dynamic discovery functionality
- Limited query capabilities: search for services restricted to WS name and its classification

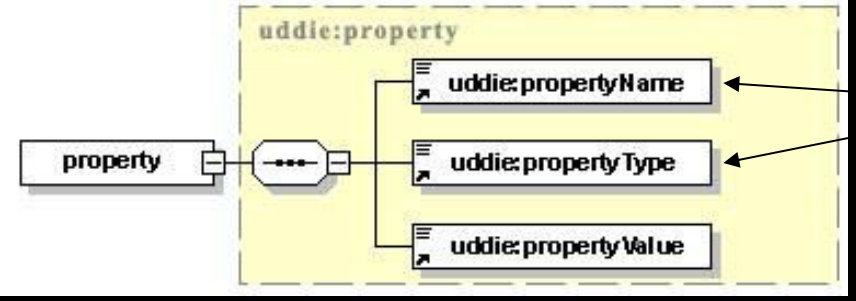
Future Extensions to UDDI

- Information on spatial and temporal availability of a service
- Information on pricing, payment and delivery channels
- Information on degree of security and confidentiality of service request
- Information on consumption, quality of service and reputation

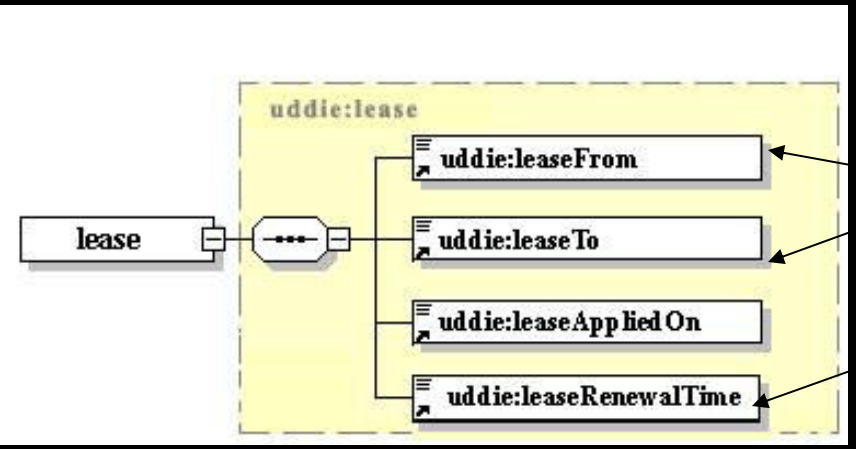
Recent research to extend UDDI Capabilities

UDDIe

- an Extension to UDDI v.2
- support the notion of "Blue Pages"
- brings the notion of leasing for registry entries:
  - dynamic service life period
- brings the notion of service properties as WS metadata
  - services may have one or more properties
  - service discovery is based on service properties
- Current status:
  - UDDIe is developed by the School of Computer Science atCardiff University.
  - Link to UDDIe procjet. http://www.wesc.ac.uk/projects/uddie/uddie/



User Defined -- may use some predefined ontology or metadata format (can be strings or number)



DD/MM/YYYY hh:mm:ss

Number of times lease renewed

UDDIe architecture

BusinessEntity

BusinessService

categoryBag
ServiceName
ServiceDescription
PropertyBag
ServiceLease

UDDIe

BindingTemplate

TModel

UDDI Resources - I

- UDDI pages, www.uddi.org
- OASIS UDDI Specifications Technical Committee manages and developes UDDI Specifications
- http://www.oasis-ope-open.org/committees/uddi-spec/index.shtml
- UDDI V2 Specifications

  http://www.oasis-open.org/committess/uddi-spec/tcspecs.shtml#uddiv2
- UDDI V3 Specifications

  http://www.oasis-open.org/committess/uddi-spec/tcspecs.shtml#uddiv3
- A recent technical note on WSDL and UDDI relationship

  http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v200-20031104.htm
- Microsoft UDDI, http://uddi.microsoft.com
- IBM UDDI,

  http://www-306.ibm.com/software/solutions/webservices/uddi
- UDDI Reference http://www.zvon.org/xxl/uddiReference/Output

UDDI Resources - II

- Development Tools
  - Microsoft UDDI SDK, For .NET framework, Windows. http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001204
  - UDDI4J, UDDI4J is a Java class library that provides an API to interact with a UDDI registry. Open source

    http://www-124.ibm.com/developerworks/oss/uddi4j/
  - jUDDI, jUDDI is a Java implementation, BSD License.

    http://freshmeat.net/projects/juddi/
  - SOAP UDDI, SOAP UDDI is a reference implementation of the UDDI specification

    http://freshmeat.net/projects/soapuddi/?topic_id=250

Shortcomings of UDDI is addressed with WS-Discovery Specs.

- UDDI provides discovery for services that are always connected to the network.
  - need a discovery system for sometimes connected services.
- UDDI has to handle with out-of-date service entry information
  - Need a discovery system for dynamically updated entries.
- UDDI provides discovery for only registered services
  - Need a discovery for services not exist in any central registry.
- UDDI provides a central registry
  - Need a discovery system which performs on distributed registries on ad hoc and managed networks

WS-Discovery

- defines a multicast protocol to locate services
- allows dynamic discovery of services in ad hoc and managed networks
  - discovery of temporarily-connected services providing interface to portable devices such as hand-helds, pocket pc, etc…
- enables discovery of resource-limited service implementations
- enables discovery of services by type and within scope
- leverages other Web service specifications for secure, reliable, transacted message delivery
- scales to a large number of endpoints, by defining a multicast suppression behavior if a service registry (discovery proxy) is available on the network.

WS-Discovery Specifications

http://ftpna2.bea.com/pub/downloads/ws-discovery.pdf

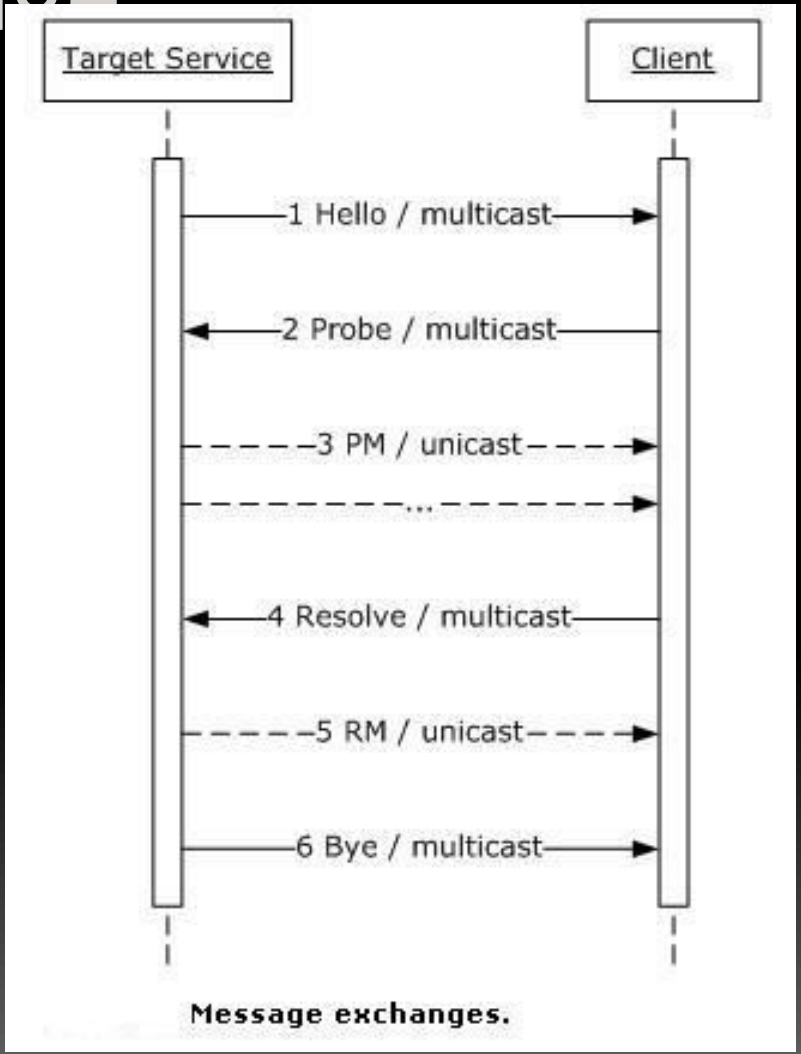
WS-Discovery

- WS-Discovery Specs. defines
  - A WSDL providing an interface for service discovery
  - A multicast discovery protocol
  - XML Schemas for WS-Discovery messages
- Current Status:
  - WS-Discovery and related specifications are provided for use as-is and for review and evaluation only.
  - Microsoft, BEA, Canon and Intel are contributors.
- Limitations
  - It does not provide liveness information on services
  - It does not define a rich data model for service description
  - It is not an internet-scale discovery

Concepts

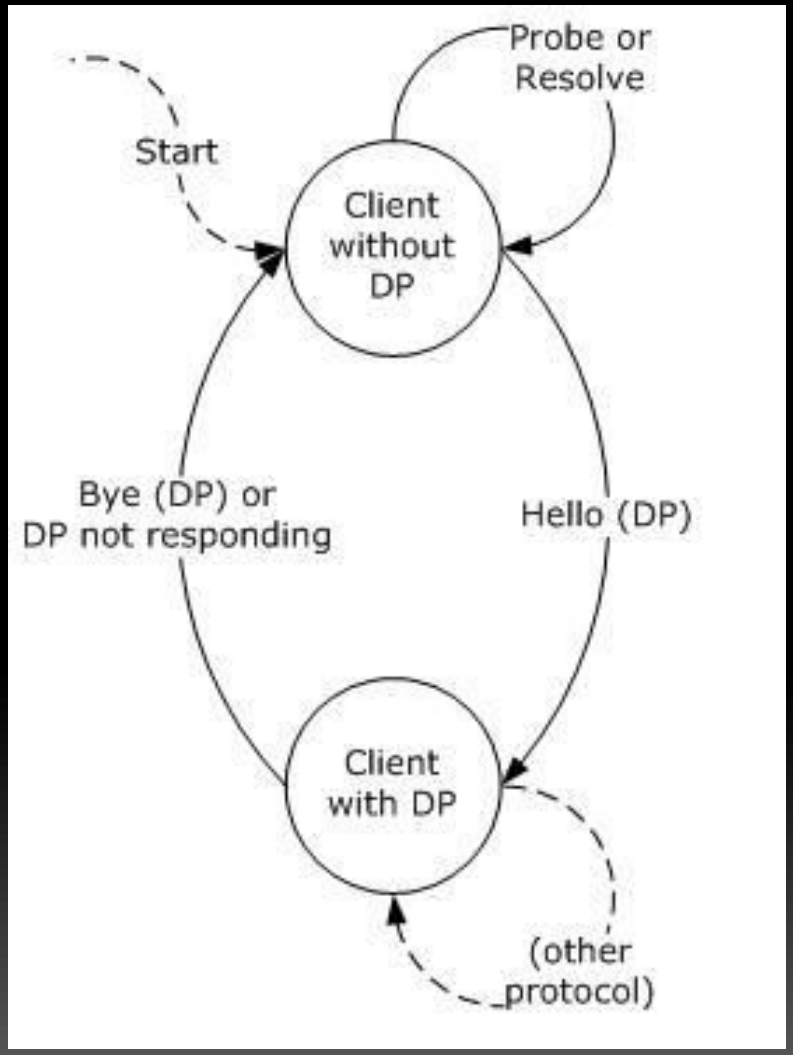
- Target Service
  - An endpoint that makes itself available for discovery.
- Client
  - An endpoint that searches for Target Service(s).
- Discovery Proxy
  - An endpoint that facilitates discovery by Clients among a large number of endpoints. Discovery Proxies are an optional component of the architecture.
- Type
  - An identifier for a set of messages an endpoint sends and/or receives (e.g., a portType).
- Scope
  - An extensibility point that may be used to organize Target Services into logical groups.
- Metadata
  - Information about the Target Service; includes, but is not limited to, network addresses where a Target Service may be reached, transports and protocols it understands, Types it implements, and Scopes it is in.

WS-Discovery Message Exchange



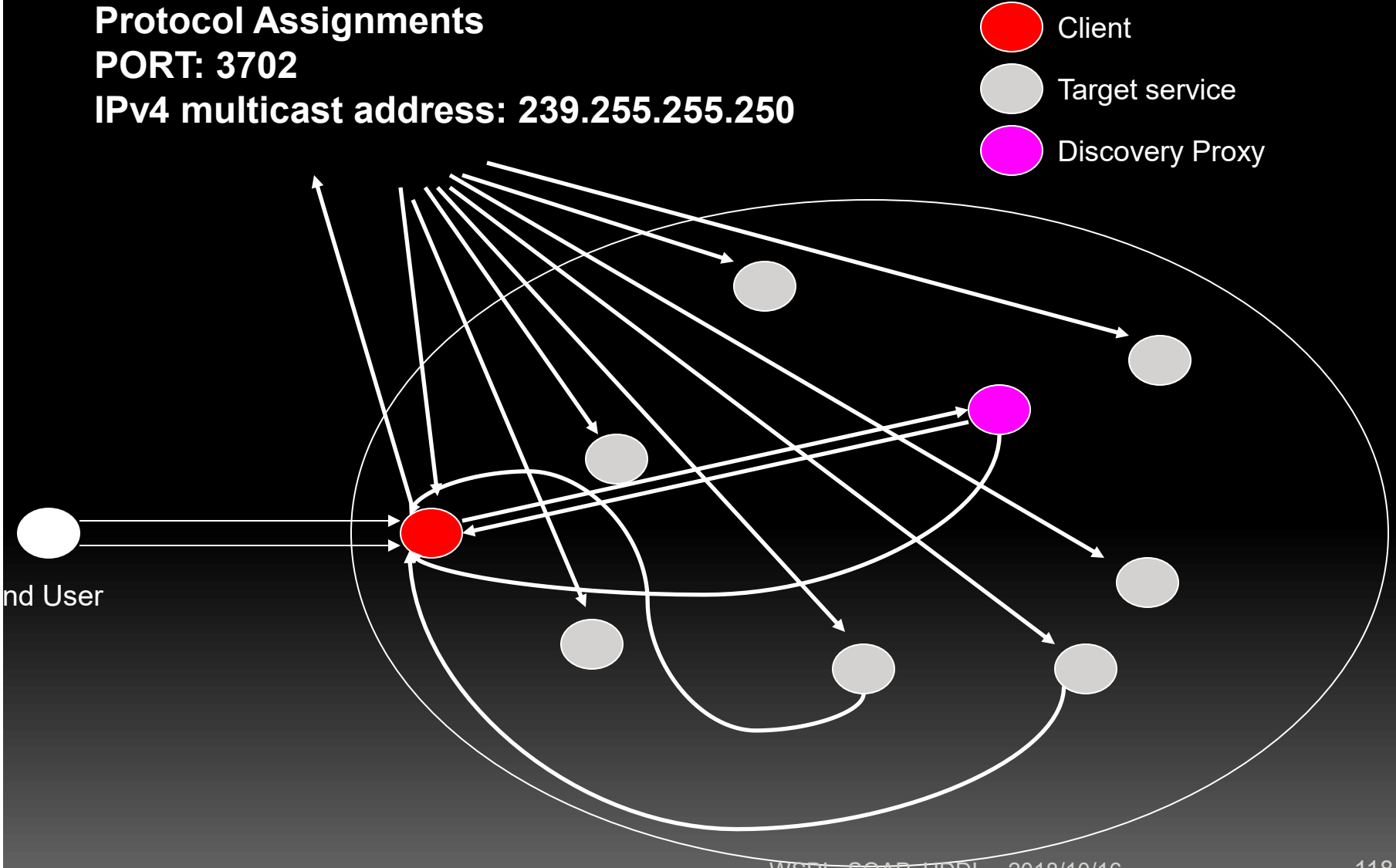
Message exchanges.

WS-Discovery Client States



WS Discovery Multicast Protocol

Protocol Assignments
PORT: 3702
IPv4 multicast address: 239.255.255.250



Discovery Proxy (D.P.)

- It is indented to be a registry for web services.
  - D.P. could be UDDI, LDAP, etc…
- When D.P. is discovered, clients use a discovery-specific protocol to communicate with one or more of them.
- WS-Discovery does not define neither the discovery-specific protocol nor the interaction between WS-Discovery service and Registry such as UDDI.
  - details are left up to the programmers.
- D.P. announces itself to the client when it detects Probe and Resolve messages. (Hello message)
- D.P. makes another announcement when it prepares to leave the system. (Bye message)

WS-Discovery Multicast Messages

- Hello
  - A message sent by a Target Service when it joins a network; the message contains key information for the Target Service.
- Bye
  - A best-effort message sent by a Target Service when it leaves a network.
- Probe
  - A message sent by a Client searching for a Target Service by Type and/or Scope.
- Resolve
  - A message sent by a Client searching for a Target Service by name.

WS-Discovery WS metadata

- Metadata includes, but is not limited to,
  - EndpointReference
  - Policy,
  - Types,
  - Scopes.
- MetadataVersion information is kept to track the changes in cached metadata
  - It is incremented by >= 1 whenever there is a change in the metadata of the Target Service.

Scope and Type of a Service

```
<xs:element name='Scope' >
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:anyURI' >
        <xs:attribute name='MatchBy' type='xs:anyURI' />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='Scopes' >
  <xs:simpleType>
    <xs:list itemType='xs:anyURI' />
  </xs:simpleType>
</xs:element>
```

Scope: anyURI

```
<xs:element name='Types' >
  <xs:simpleType>
    <xs:list itemType='xs:QName' />
  </xs:simpleType>
</xs:element>
```

Type : QName

Hello Message

- It is a one-way multicast message sent by a Target Service
- It is sent under two conditions
  - When target service joins a network
  - When metadata changes
- Hello message can be also a unicast message sent by a Discovery Proxy in response to any Probe or Resolve.
- A hello message may include metadata
  - client listens to Hello messages and stores metadata for corresponding Target Service

HELLO Message

```
<s:Envelope
 xmlns:a='http://schemas.xmlsoap.org/ws/2003/03/addressing'
 xmlns:d='http://schemas.xmlsoap.org/ws/2004/02/discovery'
 xmlns:i='http://printer.example.org/2003/imaging'
 xmlns:p='http://schemas.xmlsoap.org/ws/2002/12/policy'
 xmlns:s='http://www.w3.org/2003/05/soap-envelope' >
 <s:Header>
  <a:Action>
    http://schemas.xmlsoap.org/ws/2004/02/discovery/Hello
  </a:Action>
  <a:MessageID>
    uuid:0a6dc791-2be6-4991-9af1-454778a1917a
  </a:MessageID>
  <a:To>http://schemas.xmlsoap.org/ws/2004/02/discovery</a:To>
  <d:AppSequence InstanceId='1077004800' MessageNumber='1' />
 </s:Header>
 <s:Body>
  <d:Hello>
   <a:EndpointReference>
    <a:Address>
     uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
    </a:Address>
    <p:Policy>
     ....
    </p:Policy>
   </a:EndpointReference>
   <d:Types>i:PrintBasic i:PrintAdvanced</d:Types>
   <d:Scopes>ldap:///ou=engineering,o=examplecom,c=us</d:Scopes>
   <d:MetadataVersion>75965</d:MetadataVersion>
  </d:Hello>
 </s:Body>
</s:Envelope>
```

Metadata fields: endpoint reference, policy, type and scope of the service

Bye Message

- It is a one-way multicast message sent by a Target Service.
- It is sent when T.S. is preparing to leave the network
- Bye message can be also a unicast message sent by a Discovery Proxy when D.P. is leaving network.
- Client listens to Bye messages to invalidate cached metadata about T.S.

Bye Message

```
<s:Envelope
   xmlns:a='http://schemas.xmlsoap.org/ws/2003/03/addressing'
   xmlns:d='http://schemas.xmlsoap.org/ws/2004/02/discovery'
   xmlns:s='http://www.w3.org/2003/05/soap-envelope' >
  <s:Header>
   <a:Action>
      http://schemas.xmlsoap.org/ws/2004/02/discovery/Bye
   </a:Action>
   <a:MessageID>
      uuid:337497fa-3b10-43a5-95c2-186461d72c9e
   </a:MessageID>
   <a:To>http://schemas.xmlsoap.org/ws/2004/02/discovery</a:To>
   <d:AppSequence InstanceId='1077004800' MessageNumber='2' />
  </s:Header>
  <s:Body>
   <d:Bye>
     <a:EndpointReference>
      <a:Address>
        uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
      </a:Address>
     </a:EndpointReference>
   </d:Bye>
  </s:Body>
</s:Envelope>
```

Endpoint reference.

Note: More metadata field elements can be included in a Bye message.

Probe Message

- It is a one-way multicast message sent by a Client Service.
- It is sent If a client has not discovered any Discovery Proxies to find T.S. of a given Type and/or in a given Scope.
- It may be a unicast message, if a client knows the network address of a T.S., the Probe MAY be sent directly to that network address
- Client will listen to responses (Probe Match)
  - Client may wait for a sufficient number of responses.
  - Client may repeat the Probe several times until the Client is convinced that no further responses will be received.

Probe Message

```
<s:Envelope
    xmlns:a='http://schemas.xmlsoap.org/ws/2003/03/addressing'
    xmlns:d='http://schemas.xmlsoap.org/ws/2004/02/discovery'
    xmlns:i='http://printer.example.org/2003/imaging'
    xmlns:s='http://www.w3.org/2003/05/soap-envelope' >
  <s:Header>
    <a:Action>
      http://schemas.xmlsoap.org/ws/2004/02/discovery/Probe
    </a:Action>
    <a:MessageID>
      uuid:0a6dc791-2be6-4991-9af1-454778a1917a
    </a:MessageID>
    <a:To>http://schemas.xmlsoap.org/ws/2004/02/discovery</a:To>
  </s:Header>
  <s:Body>
    <d:Probe>
      <d:Types>i:PrintBasic</d:Types>

      <d:Scope MatchBy='http://schemas.xmlsoap.org/ws/2004/02/discovery/ldap'>
        ldap:///ou=engineering,o=examplecom,c=us
      </d:Scope>
    </d:Probe>
  </s:Body>
</s:Envelope>
```

Type

Scope

Probe Match Message

- It is a unicast message.
- It is sent by a Target Service.
- If a Target Service matches a Probe, the Target Service MUST respond with a Probe Match message.
- It may include metadata about service.

```
s:Envelope
xmlns:a='http://schemas.xmlsoap.org/ws/2003/03/addressing'
xmlns:d='http://schemas.xmlsoap.org/ws/2004/02/discovery'
xmlns:i='http://printer.example.org/2003/imaging'
xmlns:p='http://schemas.xmlsoap.org/ws/2002/12/policy'
xmlns:s='http://www.w3.org/2003/05/soap-envelope' >
<s:Header>
  <a:Action>
    http://schemas.xmlsoap.org/ws/2004/02/discovery/ProbeMatch
  </a:Action>
  <a:MessageID>
    uuid:e32e6863-ea5e-4ee4-997e-69539d1ff2cc
  </a:MessageID>
  <a:RelatesTo>
    uuid:0a6dc791-2be6-4991-9af1-454778a1917a
  </a:RelatesTo>
  <a:To> http://schemas.xmlsoap.org/ws/2003/03/addressing/role/anonymous </a:To>
</s:Header>
<s:Body>
  <d:ProbeMatch>
    <a:EndpointReference>
      <a:Address> uuid:98190dc2-0890-4ef8-ac9a-5940995e6119</a:Address>
      <p:Policy>
        <d:SoapHttpRequestReplyAddress>
          http://prn-example/PRN42/b42-1668-a
        </d:SoapHttpRequestReplyAddress>
      </p:Policy>
    </a:EndpointReference>
    <d:Types>i:PrintBasic i:PrintAdvanced</d:Types>
    <d:Scopes>
      ldap:///ou=engineering,o=examplecom,c=us
      ldap:///ou=floor1,ou=b42,ou=anytown,o=examplecom,c=us
    </d:Scopes>
    <d:MetadataVersion>75965</d:MetadataVersion>
  </d:ProbeMatch>
</s:Body>
</s:Envelope>
```

Probe Match Message

service metadata elements

Resolve Message

- It is one-way multicast message sent by a Client.
- A Client may send a Resolve message
  - If a Client has an Endpoint Reference for a T.S., and does not have enough metadata to bootstrap communication with the T.S.
  - If it has not discovered any Discovery Proxies

Resolve Message

```
<s:Envelope ...>
 <s:Header ...>
  <a:Action ...>
     http://schemas.xmlsoap.org/ws/2004/02/discovery/Resolve
  </a:Action>
  <a:MessageID ...>xs:anyURI</a:MessageID>
 [<a:ReplyTo ...>
   <a:Address ...>xs:anyURI</a:Address>
   ...
  </a:ReplyTo>]?
  <a:To ...>xs:anyURI</a:To>
  ...
 </s:Header>
 <s:Body ... />
</s:Envelope>
```

Resolve Match

- It is a unicast message.
- It is sent by a Target Service.
- If a Target Service matches a Resolve, the Target Service MUST respond with a Resolve Match message.
- It may include metadata about service.

Resolve Match Message

```
<s:Envelope ...>
  <s:Header ...>
    <a:Action ...>
       http://schemas.xmlsoap.org/ws/2004/02/discovery/ResolveMatch
    </a:Action>
    <a:MessageID ...>xs:anyURI</a:MessageID>
    <a:RelatesTo ...>xs:anyURI</a:RelatesTo>
    <a:To ...>xs:anyURI</a:To>
    ...
  </s:Header>
  <s:Body ...>
    <d:ResolveMatch ...>
      <a:EndpointReference ...>
        <a:Address ...>xs:anyURI</a:Address>
       [<a:ReferenceProperties ...>...</a:ReferenceProperties>]?
        ...
       [<p:Policy>policy expression</p:Policy>]?
      </a:EndpointReference>
     [<d:Types ...>list of xs:QName</d:Types>]?
     [<d:Scopes ...>list of xs:anyURI</d:Scopes>]?
      <d:MetadataVersion ...>xs:nonNegativeInteger</d:MetadataVersion>
      ...
    </d:ResolveMatch>
  </s:Body>
</s:Envelope>
```

Shortcomings of WS-Discovery Specifications.

- WS-Discovery does not provide liveness information on WSs.
- WS-Discovery protocol assignment is limited to a multicast address.
  - This creates dependency to multicasting system (hardware or software).
- WS-Discovery does not provide rich metadata model on WS information.
- WS-Discovery does not provide a discovery-proxy protocol for interactions between clients and registries.