Service Discovery by UDDI

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 business Entity structure

Business Key

Business Name

Services

Category

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

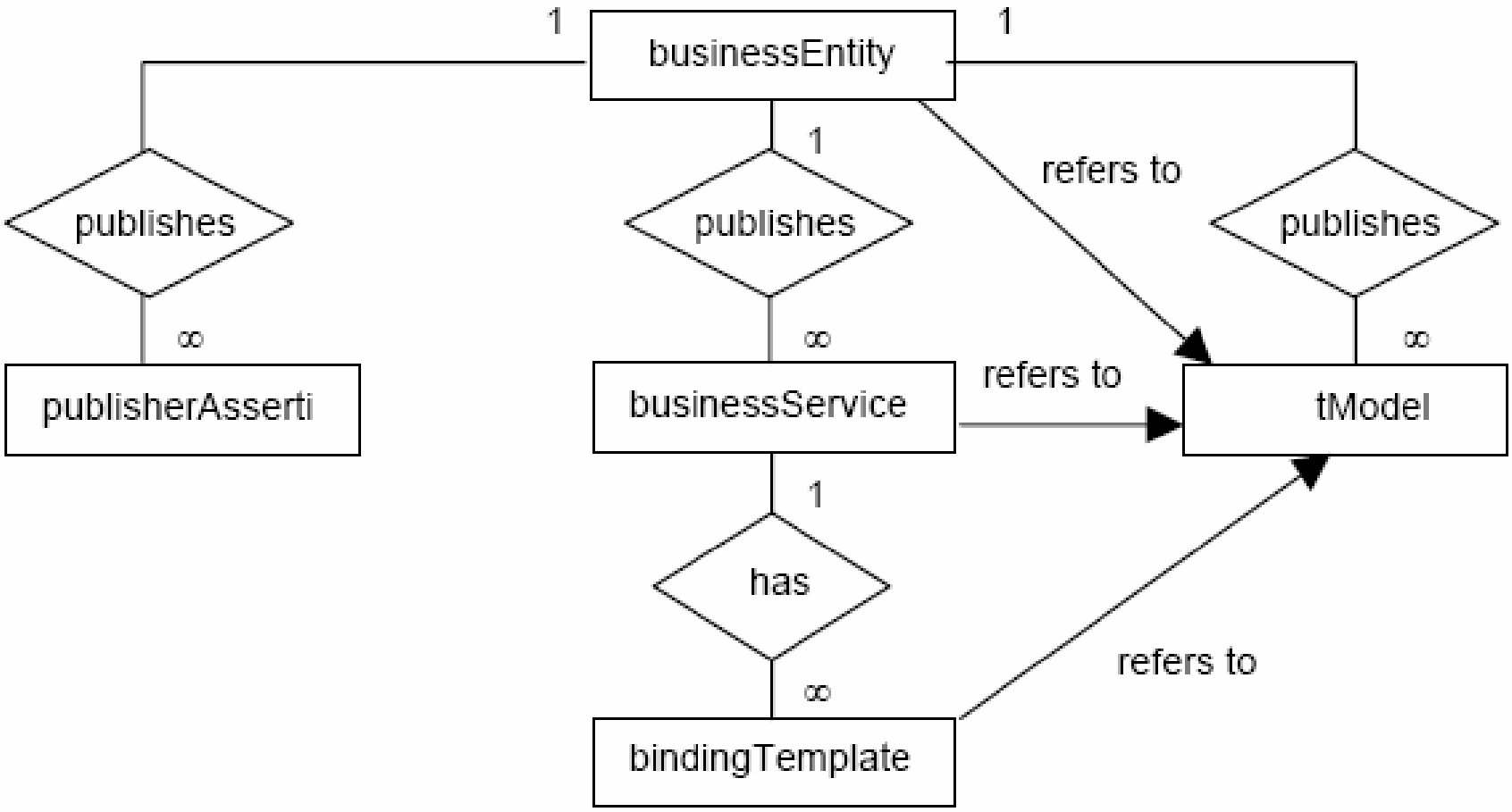
Service Name

Binding Template

tModelDetails

Category

UDDI Information Model

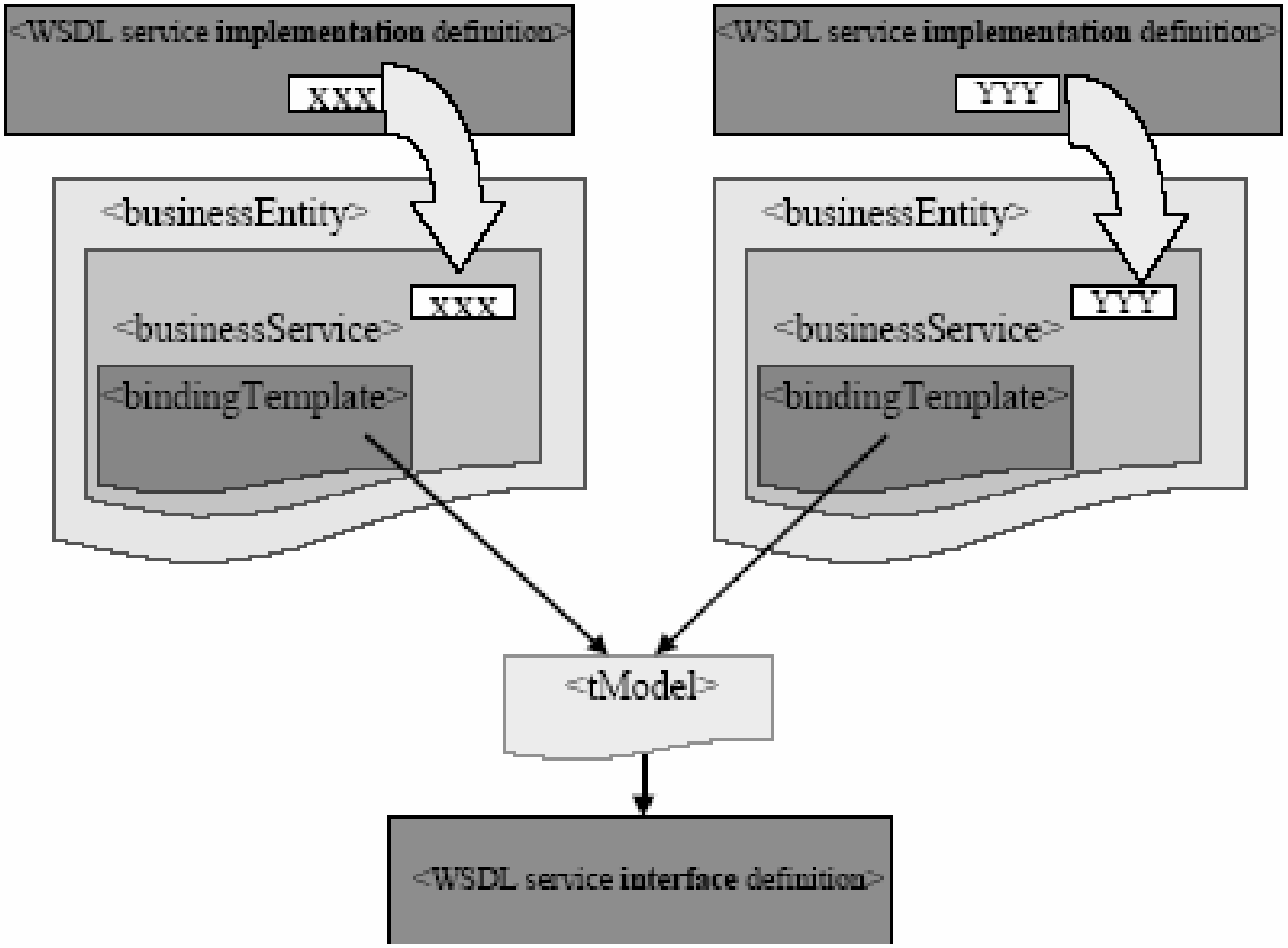


Information model of UDDI

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.

Adams, Boeyen 02

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

```
<?xml version="1.0" encoding="utf-8"?>
<definitions name="StockQuote" targetNamespace="http://example.com/stockquote/"
xmlns:tns=http://example.com/stockquote/
  xmlns:xsd1="http://example.com/stockquote/schema/" xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
  xmlns="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>
</definitions>
```

"Input\Output" Details

A sample WSDL file

input
output

- 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

```
<tModel 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>
</tModel>
```

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

```
<find_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

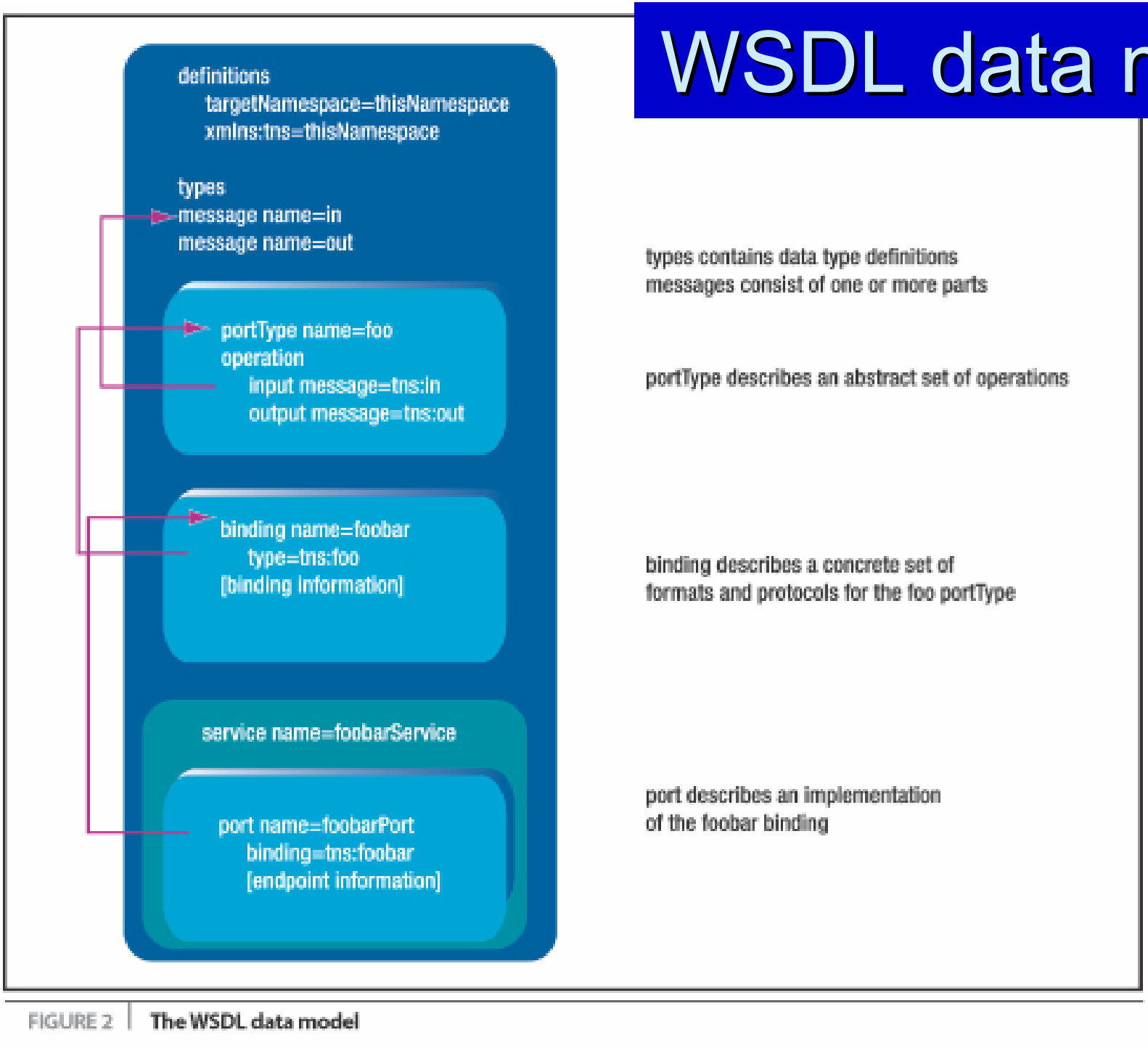


FIGURE 2 | The WSDL data model

UDDI data model

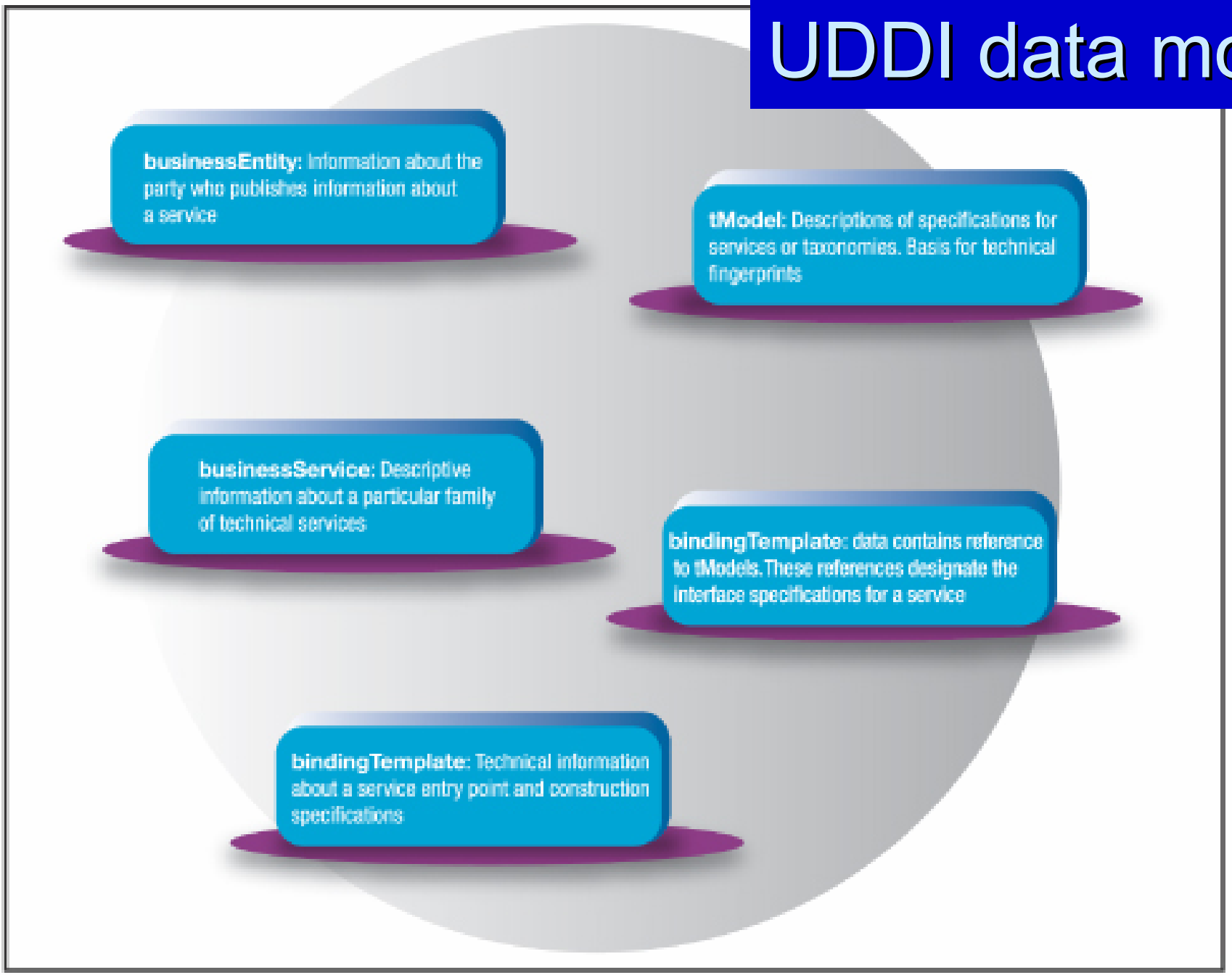
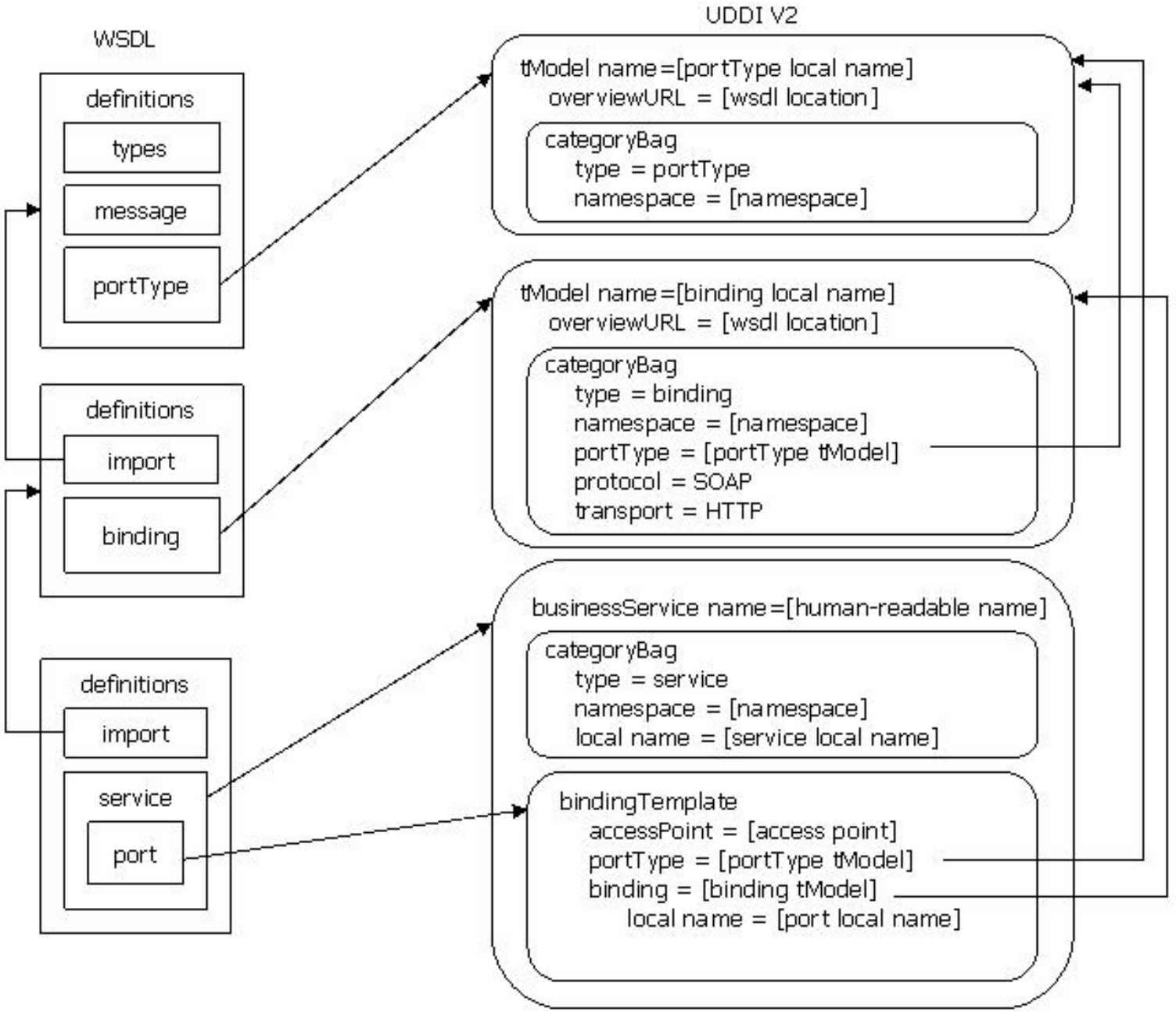


FIGURE 1 | **The UDDI data model**

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

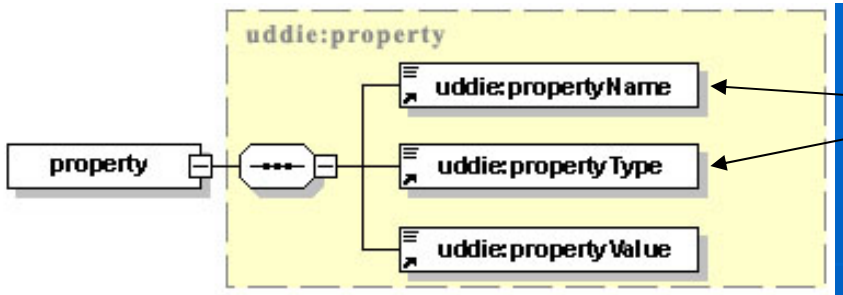
Future Extensions to UDDI

- 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

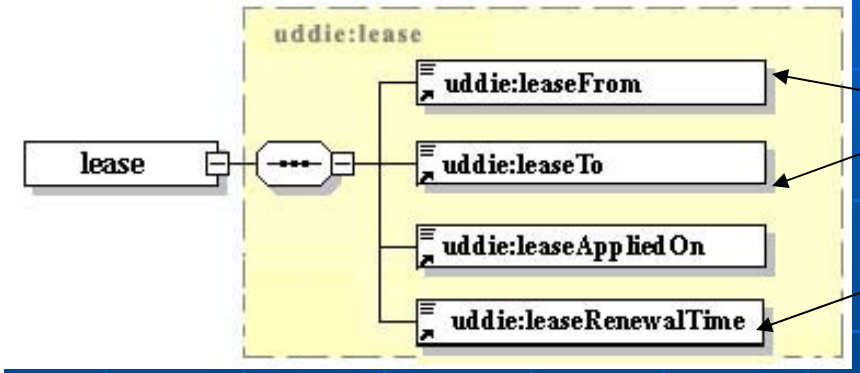
Recent research to extend UDDI Capabilities

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

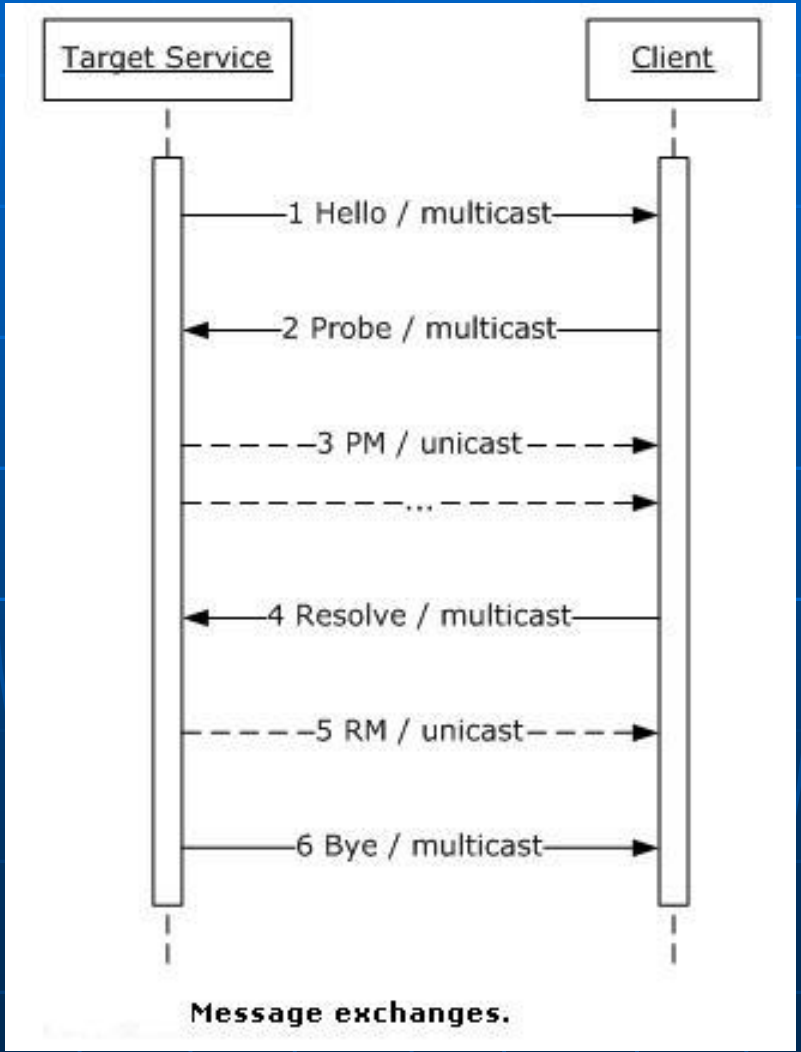
WS-Discovery

- 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

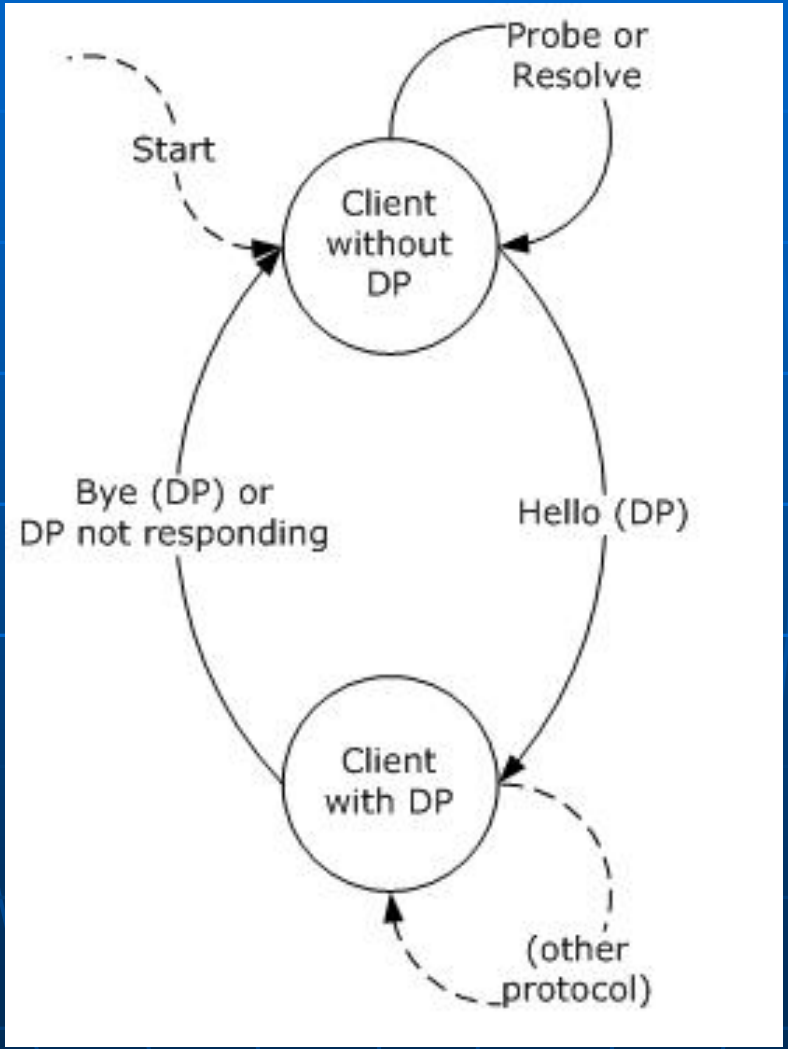
- Target Service
  - An endpoint that makes itself available for discovery.
- Client
  - An endpoint that searches for Target Service(s).
- Discovery Proxy
  - An endpoint that facilitates discovery by Clients among a large number of endpoints. Discovery Proxies are an optional component of the architecture.
- 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



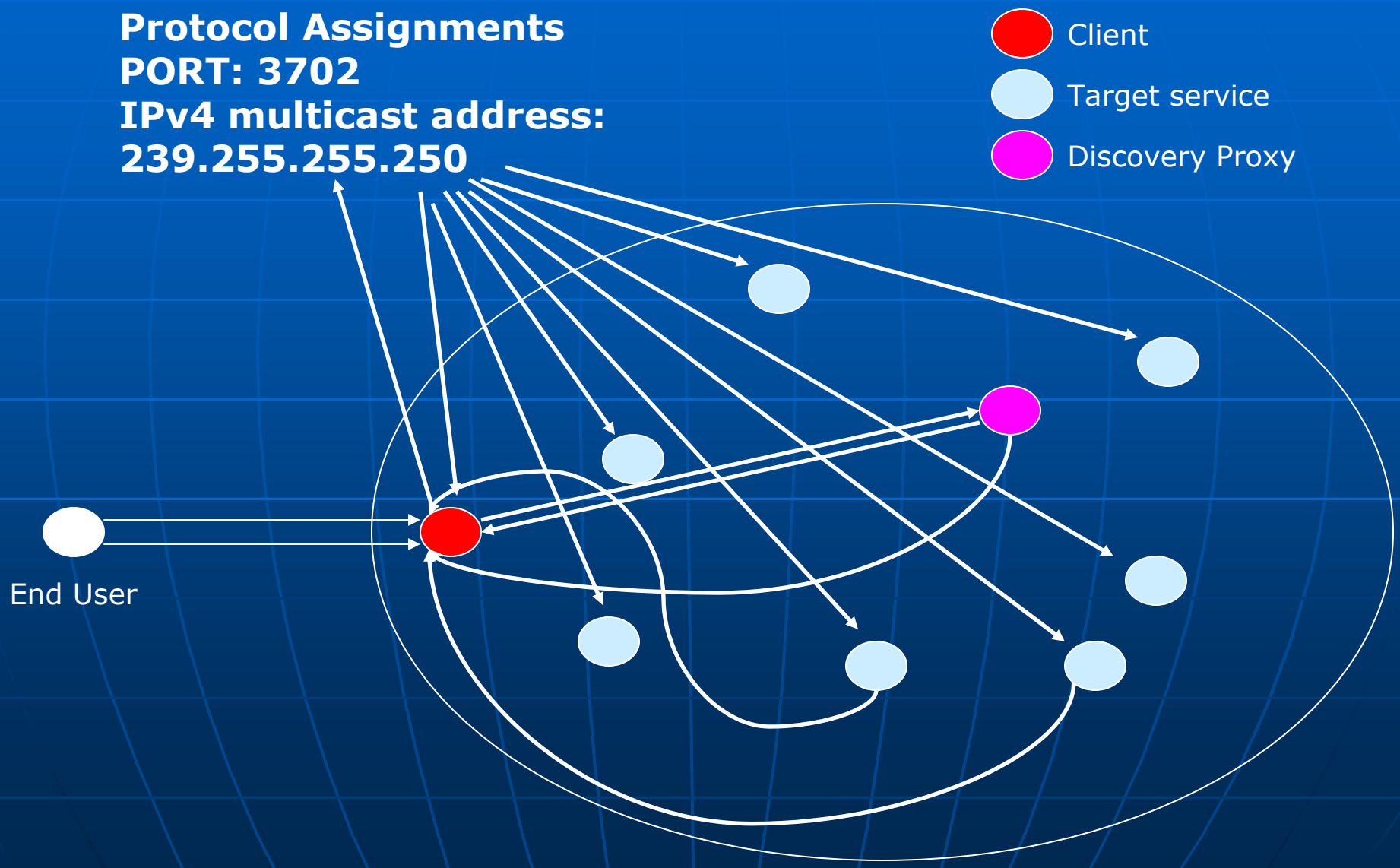
Message exchanges.

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.

Probe Match 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/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>
```

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.