Middleware and Distributed Systems

Naming and Directory Services

Martin v. Löwis

Naming

- Communication and resource sharing demands an identifier
  - Refer to locations, identify resources and other entities
- Name: string of bits or characters, used to refer to an entity (Tanenbaum)
  - Hosts, printers, files, disks, processes, users, mailboxes network connections, web pages, messages, ...
  - Entities can be operated on -> demands access point, a special entity
- Address: Name of an access point
  - Can change during the lifetime of the entity
- Client could hold address information directly
  - Interest in level of indirection, to decouple entity address changes from the consumer

Naming

- Need for entity name which is independent from its addresses
  -> *location independent name*
- Naming (name) service
  - Management and offering of relations between names and entities
    - Identification of an address or attributes for a name
    - Identification of a machine for a service
- Directory service
  - Look up entities by giving some of their attributes

Domain Name System

- Naming database for the Internet - resolve names to IP addresses
  - RFC 1035, replaced central master file downloaded by FTP
- Different name spaces, partitioned both organizationally and geographically
  - gTLD: Top-level organizational domains (com, edu, goc, mil, net, org, int)
  - ccTLD: Own domain for each country (us, uk, de, ...)
  - Maintained by IANA
- DNS client is called a *resolver*, implements UDP request/reply communication
- Multiple requests / replies can be packed in one message

DNS Zones

- Database is distributed across a network of servers, division into zones, containing multiple resource records of different type
  - Names in the domain, without sub-domain data
  - Names and addresses for at least two authoritative name servers
  - Names of name servers with authoritative data for delegated sub-domains
  - Zone management data (caching, replication cycle)
- Secondary servers download zone file from master server
- Server can cache data, but must tell client that this is non-authoritative data
- Explicit transfer of authority
  - Parent is authoritative for delegation, child is authoritative for contents

DNS Zone Example

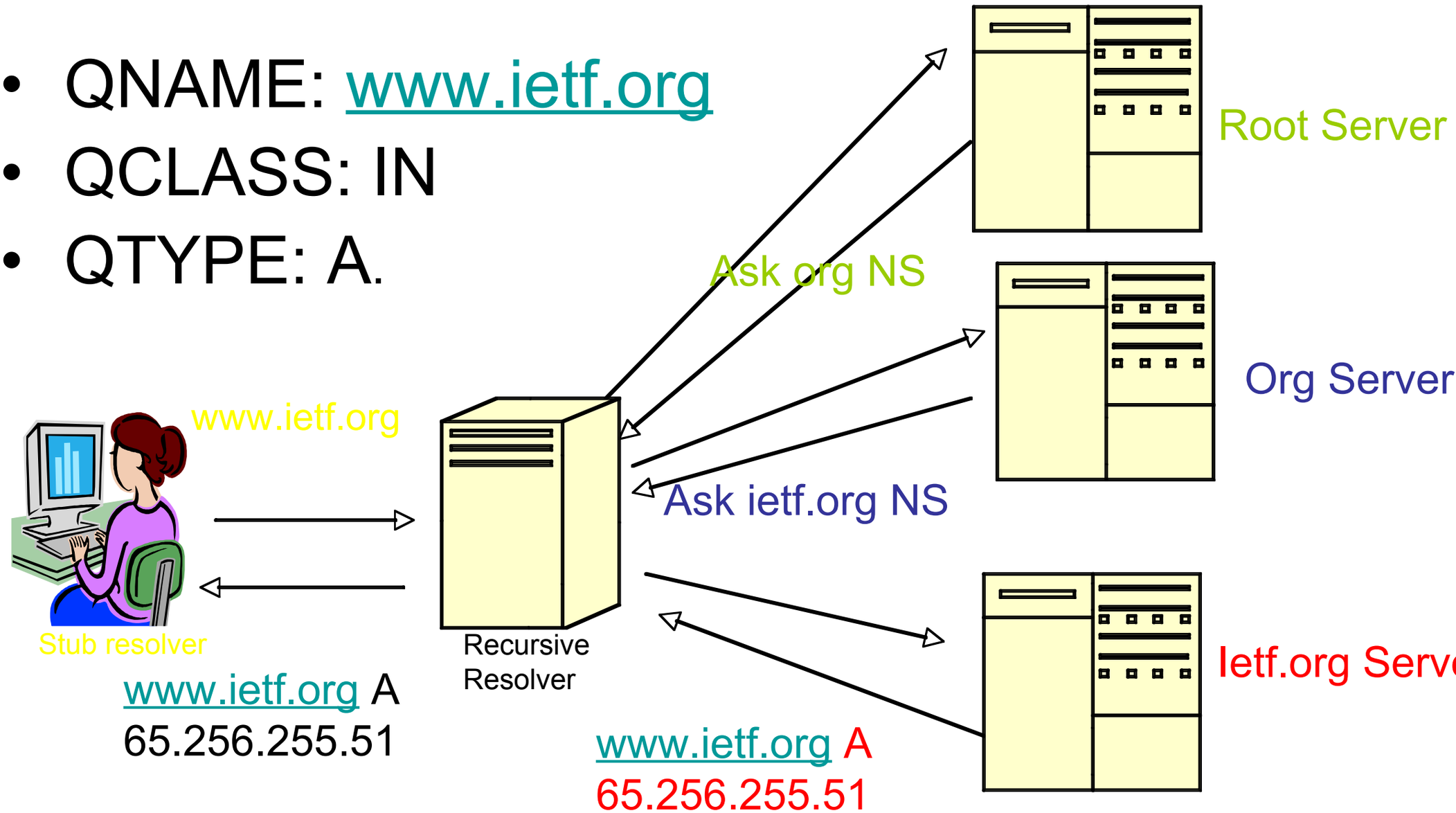
```
asg-platform.org IN SOA  dns1.hpi.uni-potsdam.de. admin.hpi.uni-potsdam.de.
(
                                2006112401 ; serial
                                28800      ; refresh (8 hours)
                                7200       ; retry (2 hours)
                                604800     ; expire (1 week)
                                172800     ; minimum (2 days)
                                )
                        NS      ns.uni-potsdam.de.
                        NS      tb0.asg-platform.org.
                        NS      dns1.hpi.uni-potsdam.de.
                        NS      schinkel.rz.uni-potsdam.de.
                        A       141.89.225.223
                        MX      10 mail2.hpi.uni-potsdam.de.
                        MX      30 mailrel.uni-potsdam.de.
bbgrid                  CNAME   tb0
docs                    A       141.89.226.3
subversion              CNAME   tb0
tb0                     A       141.89.226.2
tb2                     A       141.89.226.134
                        AAAA    2001:638:807:3:0:5efe:8d59:e286
```

DNS Queries

- Query: domain name, class (IN for Internet), type
- Domain names are case-insensitive, host names according to RFC952
- DNS query for specific type of *resource record* of a zone
  - http://www.iana.org/assignments/dns-parameters
- Host name resolution (A record)
    - Also reverse lookup possible, implemented as special zone lookup in *in-addr.arpa*
- Mail host location (MX record)
    - Query for type designation ‘mail’, result with integer preference values
- CNAME, AAAA, WKS, TXT, NS, AXFR, ...

DNS Query

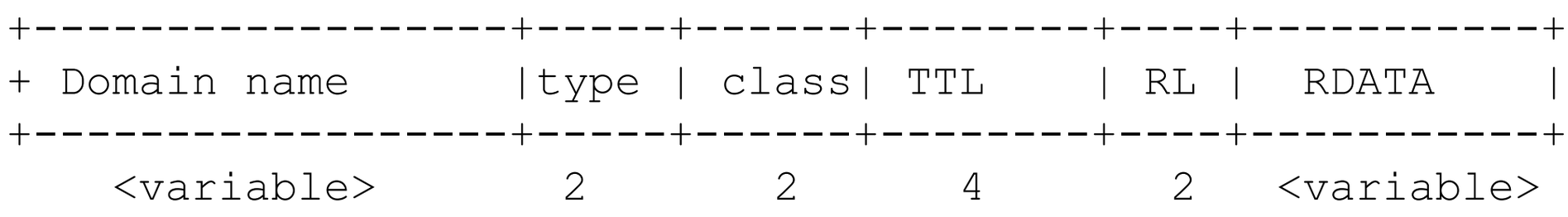
- QNAME: www.ietf.org
- QCLASS: IN
- QTYPE: A.



(C) Peter Koch

Resource Record Wire Format

- Owner name (domain name)
  - Encoded as sequence of labels, each label contains length (1 byte) and name ([1..63] bytes)
- Type : MX, A, AAAA, NS …, CLASS: IN (other classes exist but not are not globally supported)
- TTL: Time To Live in a cache
- RL: RD LENGTH: size of RDATA, RDATA: The contents of the RR
  - Binary blob, no TLV (Type Length Value).



Other DNS Related Issues

- Resource record types in the IN class
- Internationalized domain names (punycode)
- Round-robin DNS
- DNS UDP message cannot exceed 512 bytes
    - Maximum of 255 octets per name, typical maximum of 5 A records
    - Truncated flag triggers re-query over TCP
- Placing new information in DNS
    - Create new record type (talk to IANA)
    - Reuse TXT or other types

Demo

X.500

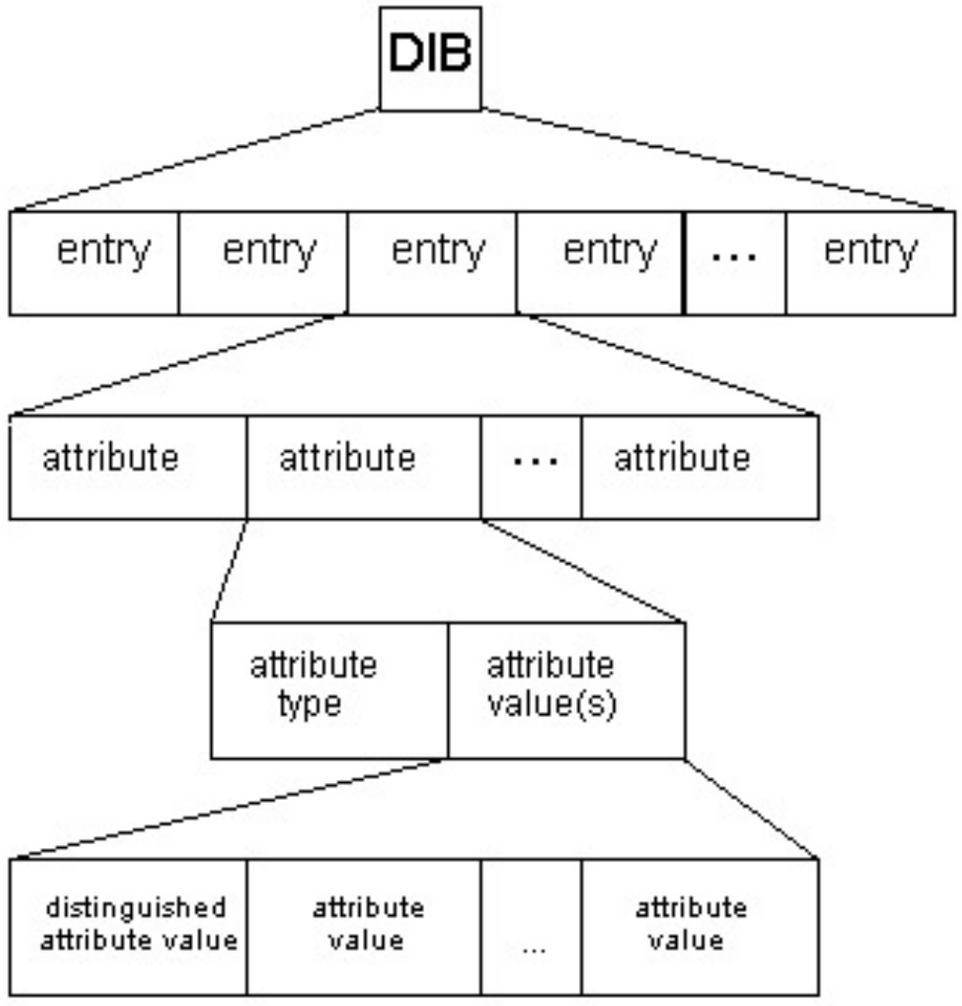
- Series of computer network standards for directory services
- CCITT / ITU-T
  - White page service for telephone numbers and addresses
- ISO /ECMA
  - Name server service for OSI applications
- Joint ISO/CCITT working group on directories (1986)
- Many on-top specifications
  - X.520 / X.521 define attributes and object classes for the description of people and organizations
  - X.509 describes authentication framework, based on X.500 directory

X.500 Models

- Directory Information Models - directory from single, global perspective
  - Directory (User) Information Model - user view
    - Information access, security, information naming
    - Some standardized information types for interoperability
  - Directory Operational and Administrative Model - admin view
    - Management of storable information types and access rights
    - Standardized set of administrative information
- Directory System Agent (DSA) Information Model
  - Information model for single computer system, to support federation
- Directory Administrative Authority Model
  - Management and administration through independent local authorities, delegation

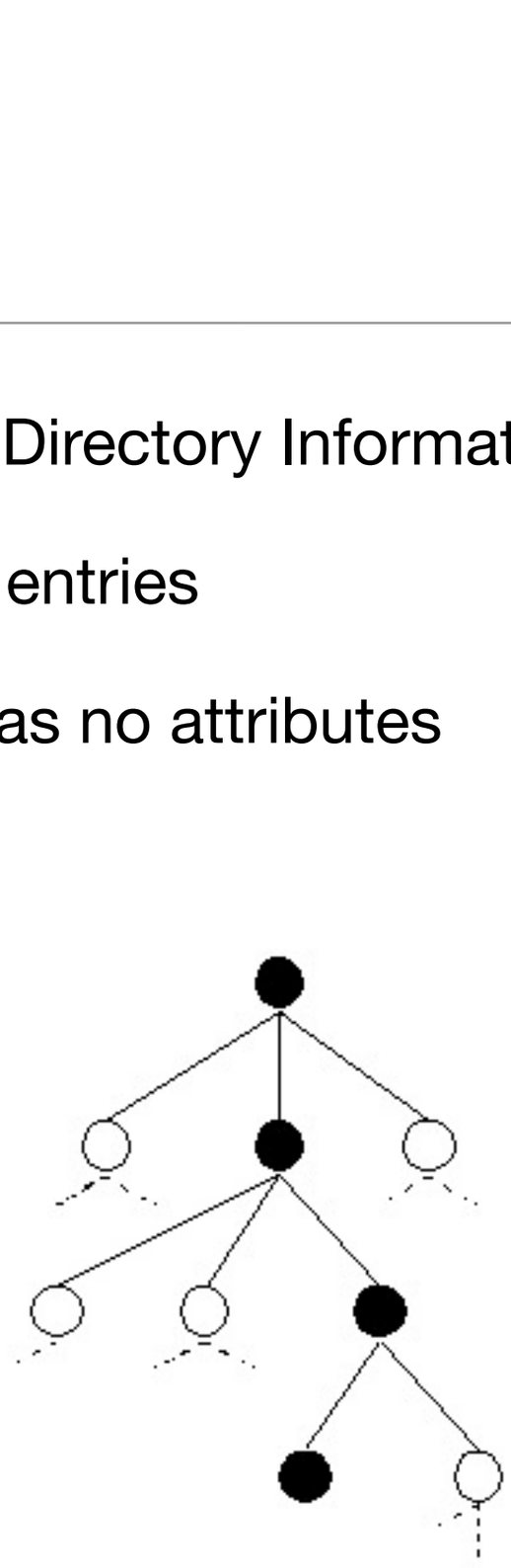
Objects and Entries

- Not all information visible to all users
  - Independent from information distribution
- Directory contains of hierarchically related *entries*
  - In Directory User Information Model: Entry holds information about object of interest to users
- One-to-one, many-to-one, one-to-many relationship to real world things
- Objects with similar characteristics are identified by their *object class*
- Some predefined attribute types



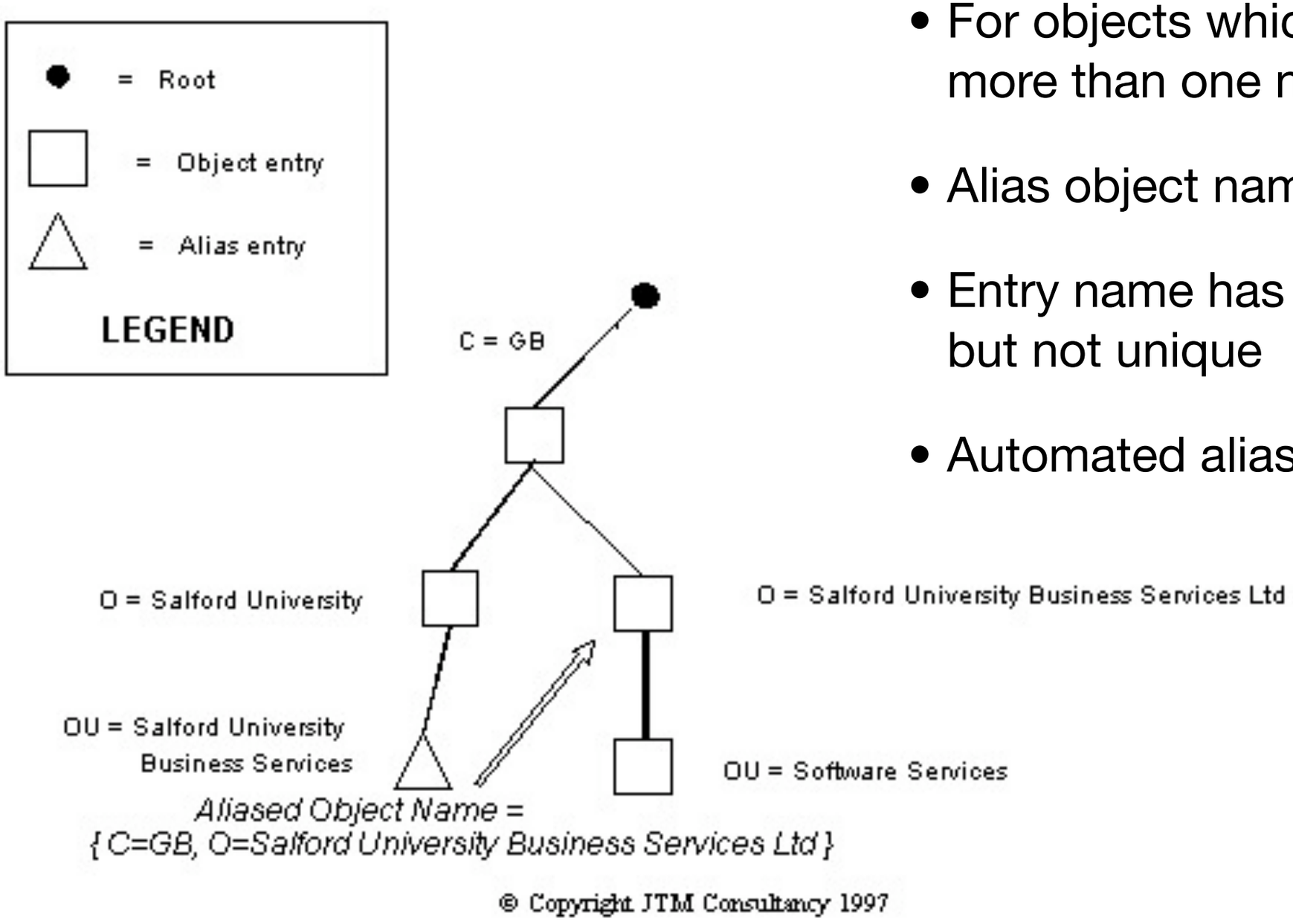
DIT

- Tree structure of entries represents Directory Information Tree (DIT)
  - Root entry, non-leaf entries, leaf entries
  - Root entry is not readable and has no attributes
  - Distributed over multiple servers
- Each object entry is unique by its *distinguished name (DN)*
  - Name of parent entry in DIT, with appended *relative distinguished name (RDN)*
  - Set of attribute type and value pairs



RDN of Entry	Distinguished Name of Entry
{null}	{null}
{Country=GB}	{Country=GB}
{Organisation=Big PLC}	{{Country=GB} Organisation=Big PLC}
{Organisational Unit=Sales,Location= Swindon}	{{{Country=GB} Organisation=Big PLC} Organisational Unit= Sales, Location= Swindon}

Aliases



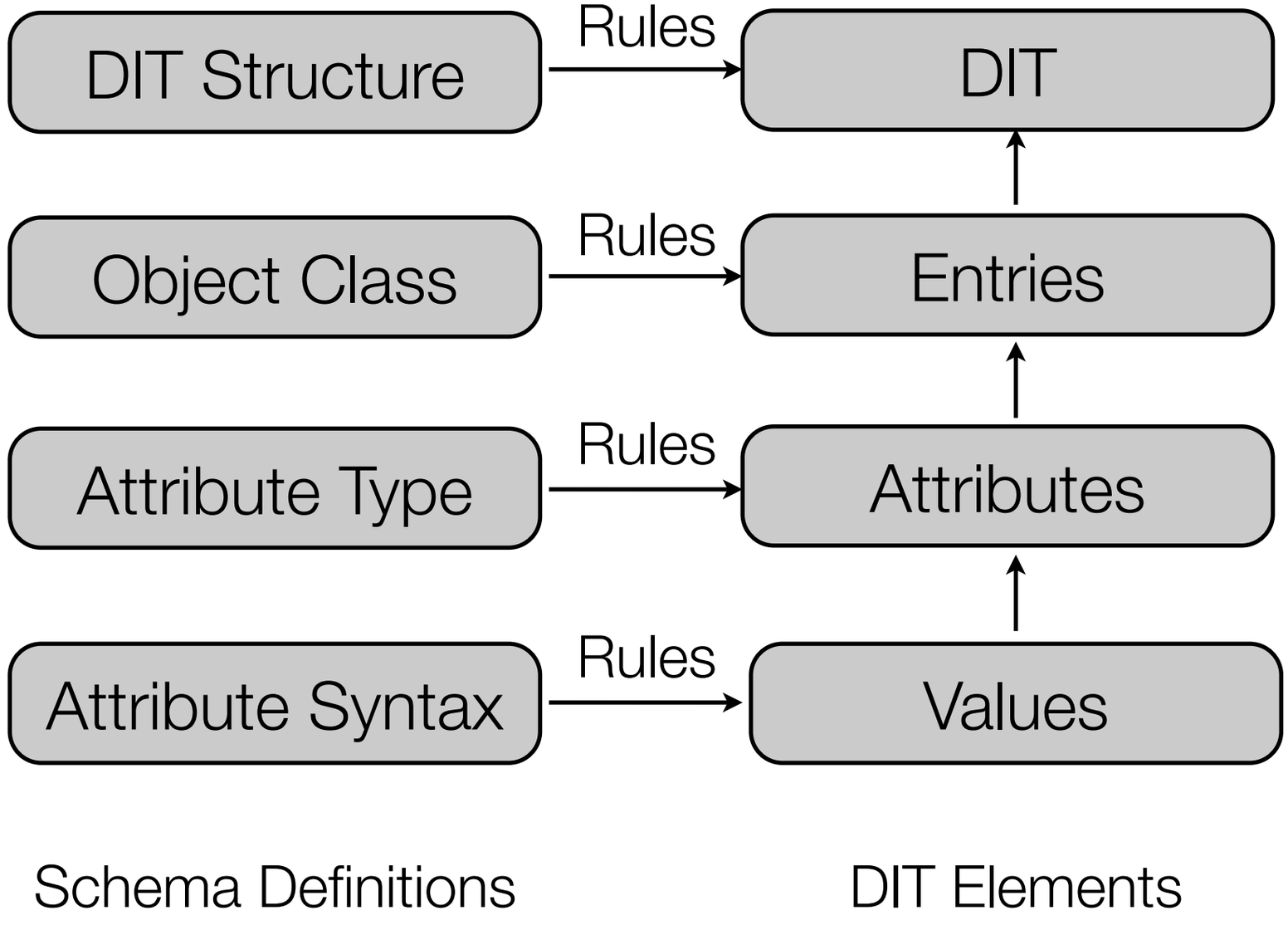
- For objects which are known by more than one name
- Alias object name attribute of entry
- Entry name has to be unambiguous, but not unique
- Automated alias dereferencing

Other X.500 Features

- Collective attributes: user attributes for a series of entries (e.g. organization phone number)
    - Different representation in User Information Model and Administrative Model (shared attribute, sub-entry)
- Attribute hierarchies (e.g. different phone numbers)
    - Search for attribute type should also return sub-type results
- Autonomous administrative areas
    - Part of directory administrative authority model
    - Authority can act as subschema administration, access control administration, collective attribute administration

X.500 Directory Schema

- Rules for controlling the storable information
- Subschemas control different portions of the directory
- Different schema components control different DIT aspects



Attribute Syntax

- Different attribute types have different syntax
  - Data type specified in ASN.1 syntax
  - Matching rules for name resolution and search operation
- Matching rules need to state
  - Attribute syntax the matching rule applies to
  - Syntax of a presented user value (might be different from attribute syntax)
  - How the comparison is performed
  - Under which conditions the match is found to be true
- Pre-defined rules, should be given for all attribute types:
  equality (for name resolution, compare, and search operation), *present* (for search), *substrings* (for search), *ordering* (for search), *approximate* (for search)

Demo

Directory Service and Protocols

- Directory Access Protocol (DAP) relays a request from a Directory User Agent (DUA) to a Directory System Agent (DSA)
- Directory System Protocol (DSP) relays user request from one DSA to another
- Bind operation
  - Connection establishment between DUA an DSA, or DSA and DSA
  - Credentials and protocol version number as parameters
  - Might not be visible for user
- Unbind operation
  - No parameters, cannot fail; similar to phone hook-up

Directory Service and Protocols

- Read operation
  - Purported name - user's best guess for the DN
  - Entry information selection
    - Select information that is returned for an entry (named attributes, all user attributes, specific attribute types, ...)
- Compare operation - Checks whether or not an entry holds a particular attribute value
- List operation - List subordinates of an entry (supported by Paged results service)
- Search operation - search through user-specified portion of DIT, select entries which meet user-defined criteria, and return selected information

LDAP

- Application-level protocol on top of TCP (RFC 2251, 1995)
  - Alternative to DAP, which relied on OSI communication stack
- Simplification of OSI access protocol for X.500 directories
  - Parameters of lookup and update operations can be passed as strings, instead of separate ASN.1 encoding
- Widely adopted for internet services (e.g. Microsoft Active Directory)
- Different operations on TCP port 389
  - *StartTLS, Bind, Search, Compare, Unbind*
  - Maintenance operations: adding, modification and deletion of entries
- LDIF: LDAP Data Interchange Format for content and update requests

LDAP Search and Compare

- *baseObject*: The DN (Distinguished Name) of the entry at which to start the search,
- *scope*: *BaseObject* , *singleLevel* (entries immediately below the base DN), or *wholeSubtree* (the entire subtree starting at the base DN)
- *filter*: How to examine each entry in the scope
  - *(&(objectClass=person)(\|(givenName=John)(mail=john*)))*
- *derefAliases*: Whether and how to follow alias entries
- *attributes:* Which attributes to return in result entries
- *sizeLimit, timeLimit*: Max number of entries, and max search time.
- *typesOnly*: Return attribute types only, not attribute values.

LDIF Example

Distinguished Name of Entry

Entry RDN | Parent Entry DN

Entry Attributes

```
dn: cn=John Doe,dc=example,dc=com
 cn: John Doe
 givenName: John
 sn: Doe
 telephoneNumber: +1 888 555 6789
 telephoneNumber: +1 888 555 1234
 mail: john@example.com
 manager: cn=Barbara Doe,dc=example,dc=com
 objectClass: inetOrgPerson
 objectClass: organizationalPerson
 objectClass: person
 objectClass: top
```

Discussion: Design of a TWP Naming Service

CORBA Naming Service

- Version 1.3, October 2004, first version 1998
- *Name binding*: Name-to-object association
  - Always relative to a *naming context,* which ensures name uniqueness
- *Name resolution:* Determine object with given name in given context
- Context itself is also an object -> *naming graph*
  - Compound name (== path) references an object (== node) in the graph
  - *Names* as sequence of *name components*, which name a context or the *bound object (*last component in the compound name)
  - Name component consists of *id* and *kind* (distinguish related objects)
- Needs *initial naming context*

Naming Service Design Principles

- No interpretation or semantics for the names themselves
- Distributed heterogeneous implementation and administration
- Name service clients need not to be aware of physical distributed infrastructure
- Fundamental object service, no dependencies to other services
- Collaboration with arbitrary and unknown implementations

Structures

- NameComponent structure
  - ISO Latin-1 strings, case-sensitive
  - Implementation can define restrictions (length, character set)
- Name
  - Non-empty sequence of *NameComponents*, length limit by application
- Binding
  - Expresses a name bound to an entity, either context or object

Modules and Interfaces

```
module CosNaming {
   typedef string Istring;    // for internationalized string type
   struct NameComponent {
      Istring id;
      Istring kind;
   }
   typedef sequence<NameComponent> Name;
   enum BindingType { nobject, ncontext };
   struct Binding {
      Name binding_name;    // should be NameComponent, always length 1
      BindingType binding_type;
   };
   typedef sequence <Binding> BindingList;
   interface BindingIterator;
   interface NamingContext {...};
   interface BindingIterator {...};
   interface NamingContextExt: NamingContext {...};
};
```

NamingContext operations

- Binding operations name an object in its naming context
  - Create new binding with *bind*
  - Bind existing name to new object with *rebind*
  - Bind name to naming context object with *bind_context*
    - Create new naming context object with *new_context* - interface implementation is then located on the naming server side, and not on the client side
    - *bind_new_context* as shortcut for naming context creation and binding
- Once an object is bound, it can be resolved by *resolve* operation
  - Takes *Name* as argument, resolution can traverse multiple contexts
- Exceptions: *NotFound, AlreadyBound, CannotProceed*, *InvalidName*, *NotEmpty*
- Rebinding allows binding even if the name is already used -> possible orphan tree with *rebind_context*
- Browse hierarchy with list operation, starting from own naming context

NamingContext

```
interface NamingContext {
   enum NotFoundReason {missing_node, not_context, not_object};
   exception ...;
   void bind(in Name n, in Object obj)
      raises( NotFound, CannotProceed, InvalidName, AlreadyBound );
   void rebind(in Name n, in Object obj)
      raises(NotFound, CannotProceed, InvalidName);
   void bind_context(in Name n, in NamingContext nc)
      raises(NotFound, CannotProceed, InvalidName, AlreadyBound );
   void rebind_context(in Name n, in NamingContext nc)
      raises(NotFound, CannotProceed, InvalidName);
   Object resolve (in Name n)
      raises(NotFound, CannotProceed, InvalidName);
   void unbind(in Name n)
      raises(NotFound, CannotProceed, InvalidName);
   NamingContext new_context();
   NamingContext bind_new_context(in Name n)
      raises( NotFound, AlreadyBound, CannotProceed, InvalidName);
   void destroy() raises(NotEmpty);
   void list( in unsigned long how_many, out BindingList bl,
              out BindingIterator bi);
```

BindingIterator and NamingContextEx

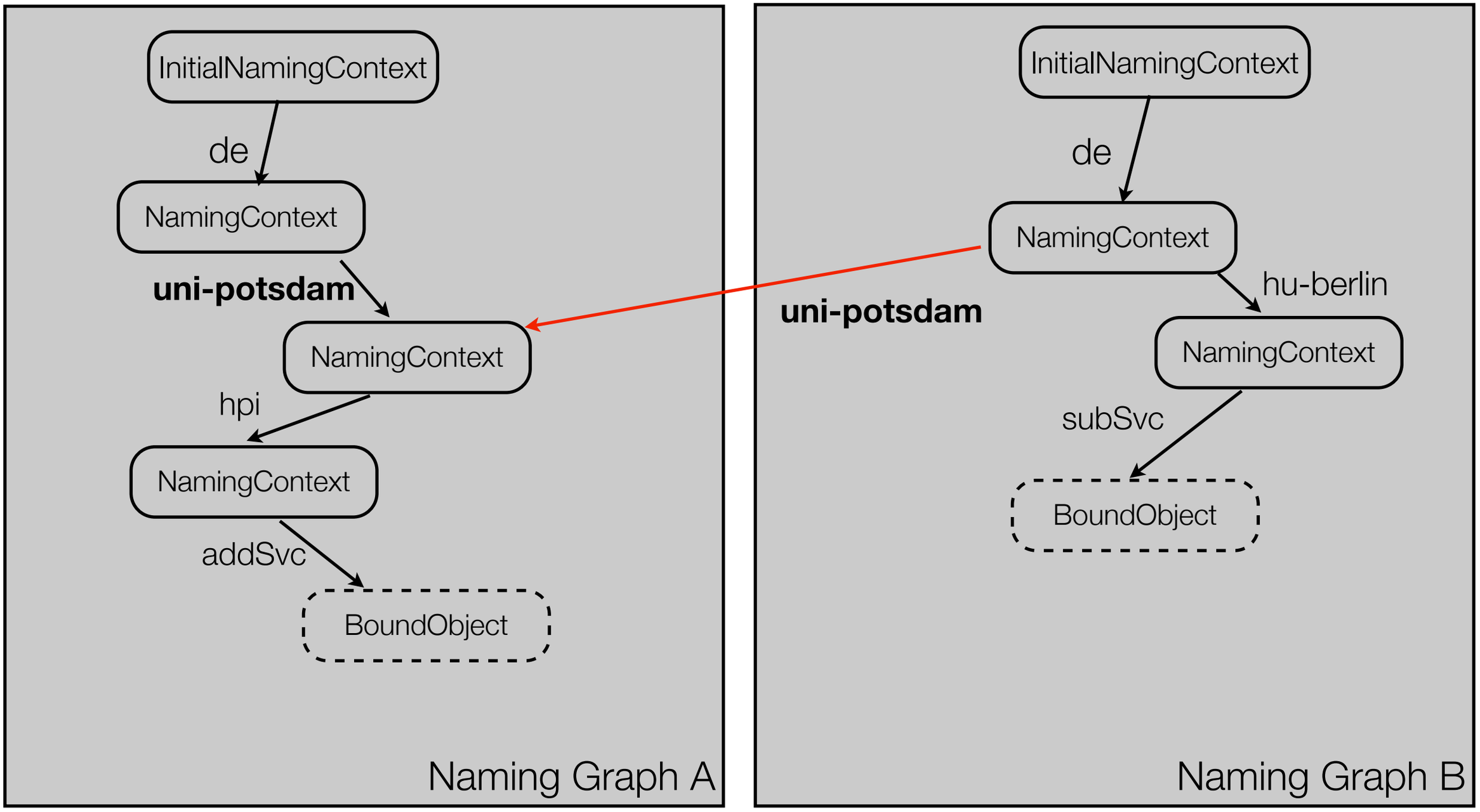
```
interface BindingIterator {
   boolean next_one(out Binding b);
   boolean next_n(in unsigned long, how_many, out BindingList bl);
   void destroy();
};

interface NamingContextExt: NamingContext {
   typedef string StringName;
   typedef string Address;
   typedef string URLString;
   StringName to_string(in Name n)
      raises(InvalidName);
   Name to_name(in StringName sn)
      raises(InvalidName);
   exception InvalidAddress {};
   URLString to_url(in Address addr, in StringName sn)
      raises(InvalidAddress, InvalidName);
   Object resolve_str(in StringName sn)
      raises( NotFound, CannotProceed, InvalidName );
};
```

Stringified Names

- Names are sequences of name components
  - Need string representation for humans, I/O activitites, ...
- Name components separated by ‘/’, escape character ‘\’
- ‘.’ to separate *id* and *kind* fields, without interpreted as *id* field only, name component with empty fields is possible
  - a.b/c.d/. or a/./c.d/.e
- URL schemes allow to represent a CORBA object
  - corbaname::555xyz.com/dev/NContext1#a/b/c
  - corbaname:rir:#a/b/c (relies on default key “NameService”)
- Interface *NamingContextExt* converts between CosNames, Strings and URLs

Federation


InitialNamingContext
de
NamingContext
uni-potsdam
NamingContext
hpi
NamingContext
addSvc
BoundObject
Naming Graph A
InitialNamingContext
de
NamingContext
hu-berlin
uni-potsdam
NamingContext
subSvc
BoundObject
Naming Graph B

UDDI

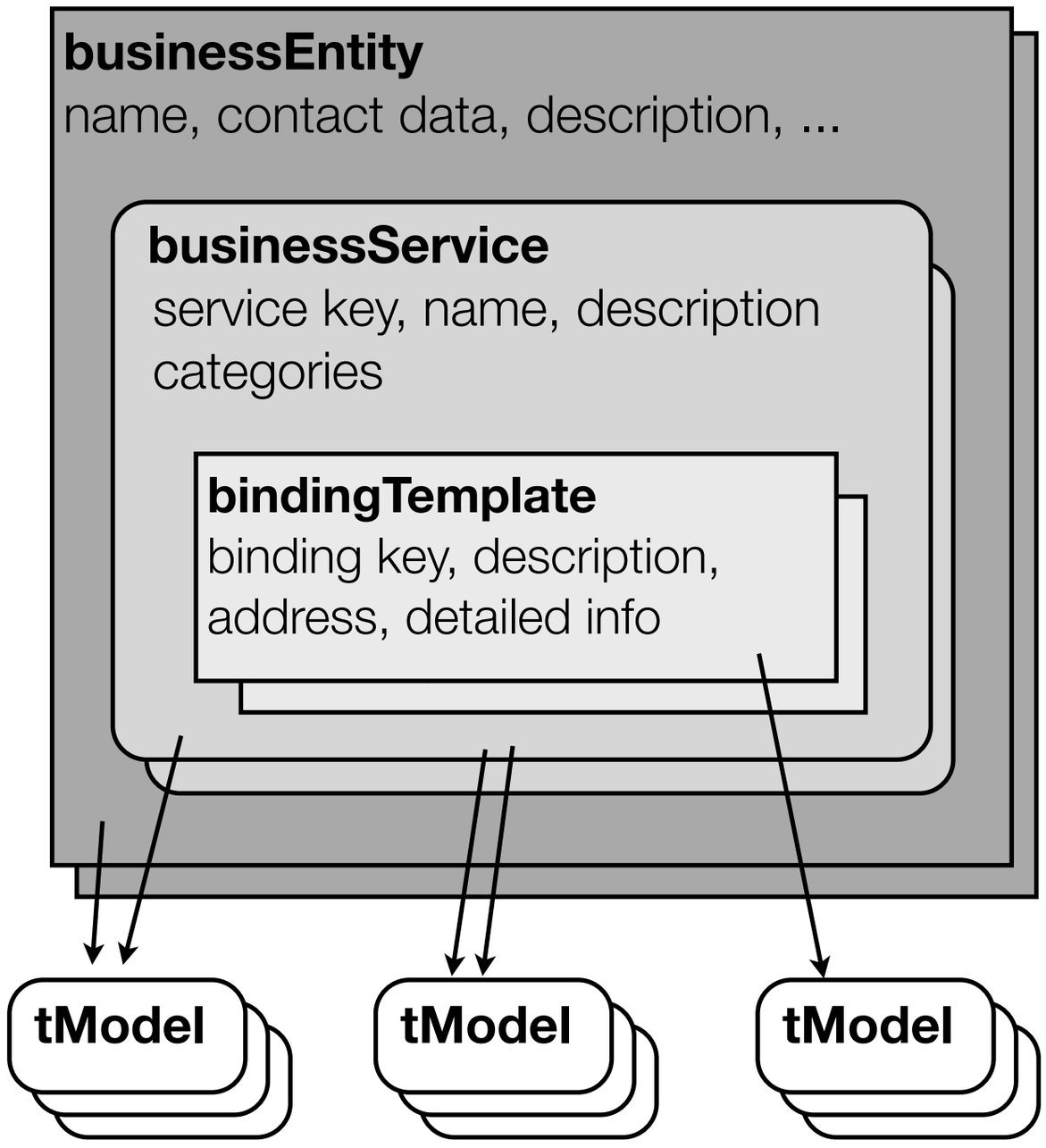
- Universal Description Discovery and Integration specification
  - First specification by IBM, Microsoft, and Ariba in 2000
  - Current version 3 (2002), maintained at OASIS
  - Intended as worldwide directories - meanwhile focus on private registries
- Framework for describing and discovering Web Services
  - “Business registry” vs. “Directory service for Web Services”
  - Specified in WSDL - a Web Service in itself

UDDI Capabilities

- A UDDI registry contains three kinds of information:
  - White pages: Information such as the name, address, telephone number, and other contact information of a given business
  - Yellow pages: Information that categorizes businesses. This is based on existing (non-electronic) standards
  - Green pages: Technical information about the Web services provided by a given business
-

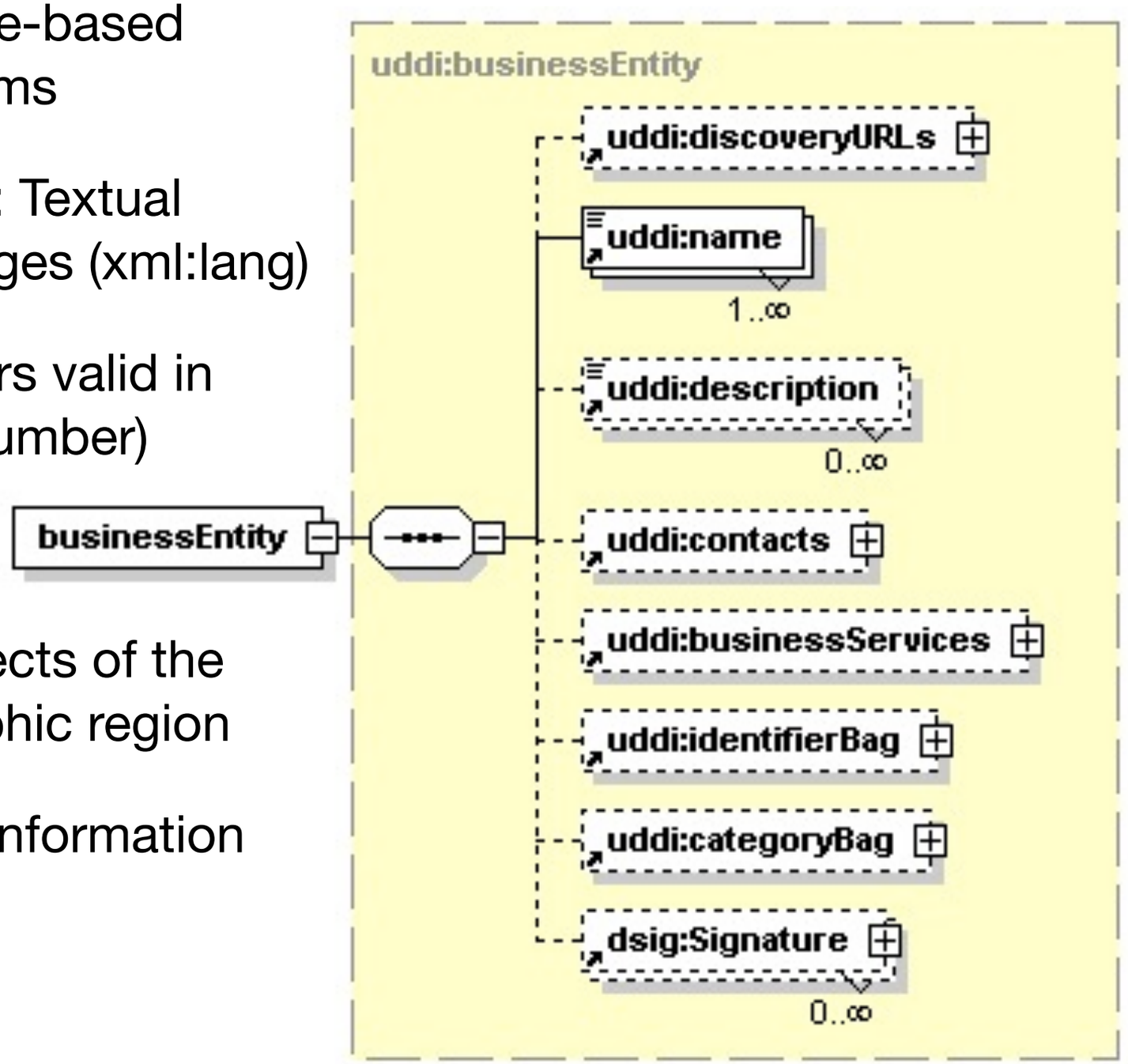
UDDI Registry Entities

- *businessEntity :* Describes organization that provides services (white pages)
- *businessService :* Describes group of related technical services with one business functionality, provided by *businessEntity* (yellow pages)
- *bindingTemplate :* Technical information to use a particular service (green pages)
  - Reference to *technical model (tModel)* describing the service properties, interface and operation parameters



businessEntity

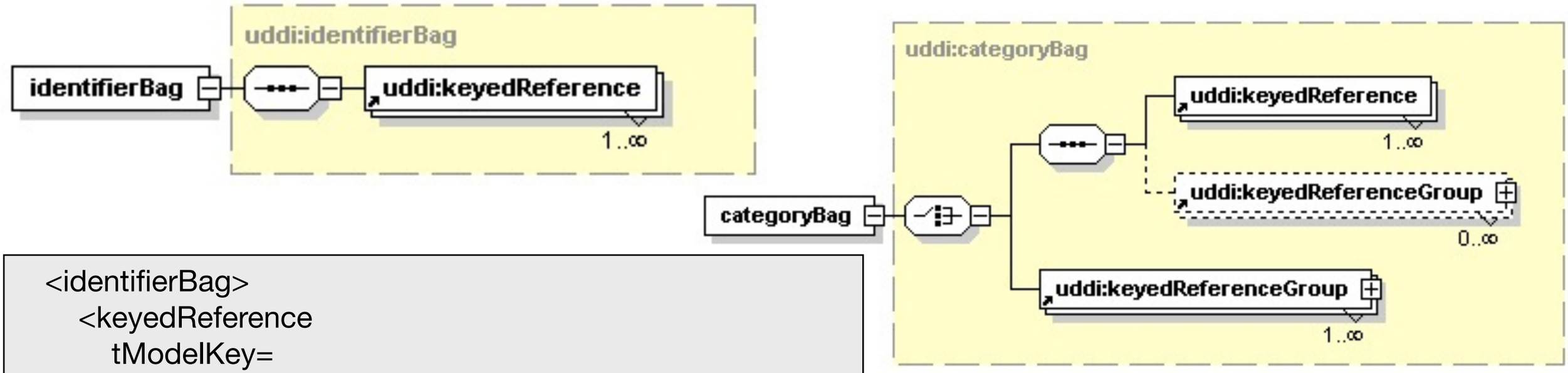
- *discoveryURLs*: alternate, file-based service discovery mechanisms
- *name, description, contacts*: Textual information, multiple languages (xml:lang)
- *identifierBag*: Other identifiers valid in their own system (e.g. tax number)
- *categoryBag*: Business aspects of the entity, e.g. industry, geographic region
- Possibility for signed entity information



businessEntity Example

```
<?xml version="1.0" encoding="utf-8"?>
<businessEntity businessKey="…" xmlns="urn:uddi-org:api_v2">
  <name>Company</name>
    <categoryBag>
      <keyedReference
         tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"
         keyName="NAICS: Software Publisher"
         keyValue="51121" />
      <keyedReference
         tModelKey="uuid:4e49a8d6-d5a2-4fc2-93a0-0411d8d19e88"
         keyName="California"
         keyValue="US-CA" />
    </categoryBag>
</businessEntity>
```

Bags




```
<identifierBag>
    <keyedReference
        tModelKey=
"uddi:uddi.org:ubr:identifier:dnb.com:d-u-n-s"
        keyName="SAP AG"
        keyValue="31-626-8655" />
</identifierBag>
```

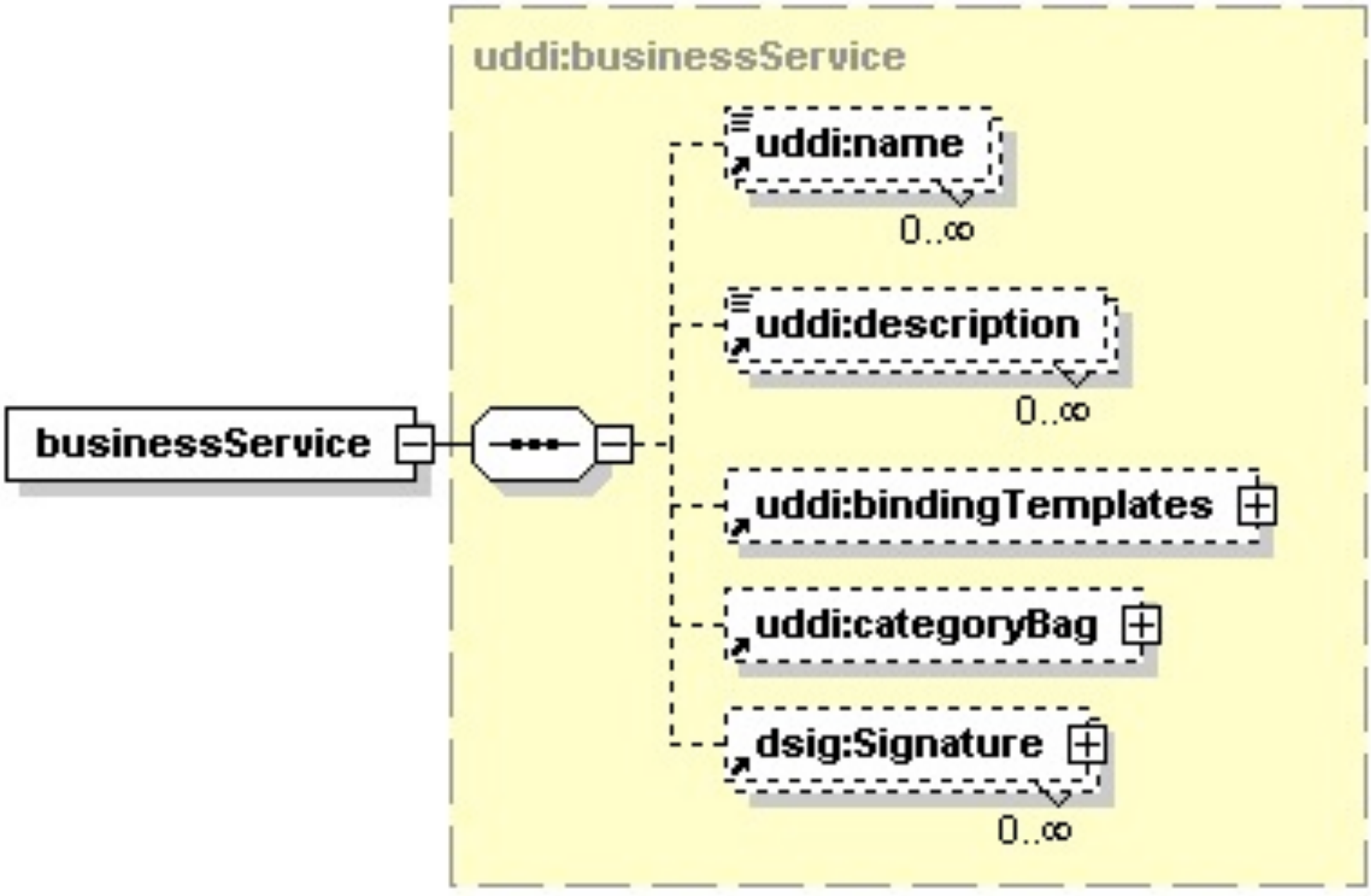
```
<keyedReferenceGroup tModelKey="uddi:uddi.org:ubr:categorizationGroup:wgs84" >
    <keyedReference
        tModelKey="uddi:uddi.org:ubr:categorization:wgs84:latitude"
        keyName="WGS 84 Latitude"
        keyValue="+49.682700" />
     <keyedReference
        tModelKey="uddi:uddi.org:ubr:categorization:wgs84:longitude"
        keyName="WGS 84 Longitude"
        keyValue="+008.295200" />
</keyedReferenceGroup>
```

KeyedReference

- In *identifierBags*
  - Represents an identifier of a specific identifier system
- In *categoryBags*
  - *tModelKey* refers to the *tModel* that represents the categorization system
  - *keyValue* contains the actual categorization within this system
- Example: Categorize business entity as offering goods in California

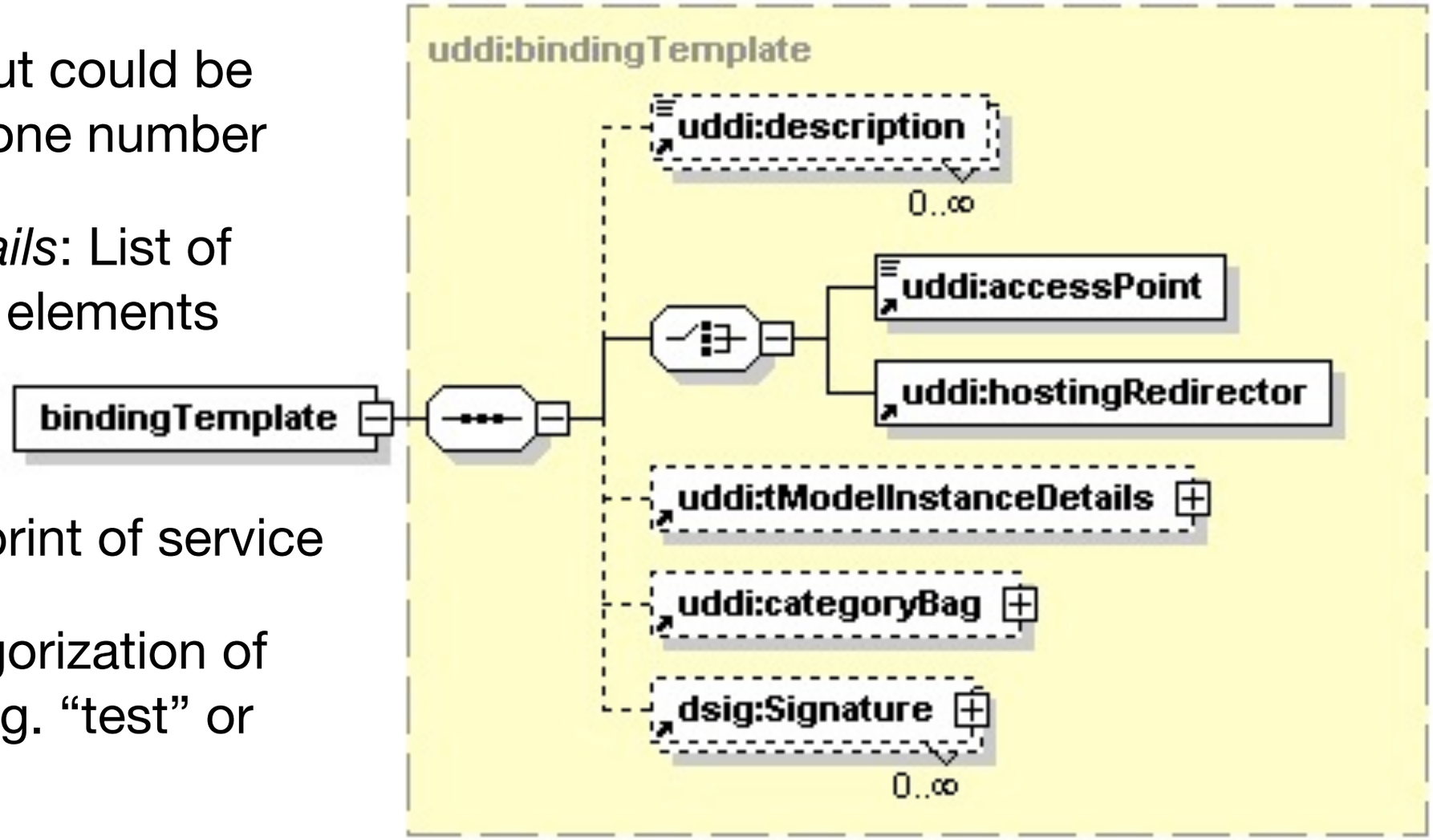
```
<keyedReference
    tModelKey="uddi:uddi.org:ubr:categorization:iso3166"
    keyName="California, USA"
    keyValue="US-CA" />
```

businessService



bindingTemplate

- *accessPoint*: Network address for service invocation
  - Typically URL, but could be also eMail or phone number
- *tModelInstanceDetails*: List of tModelInstanceInfo elements
  - Technical fingerprint of service
- *categoryBag*: Categorization of bindingTemplate, e.g. “test” or “production”
- Support for signed information

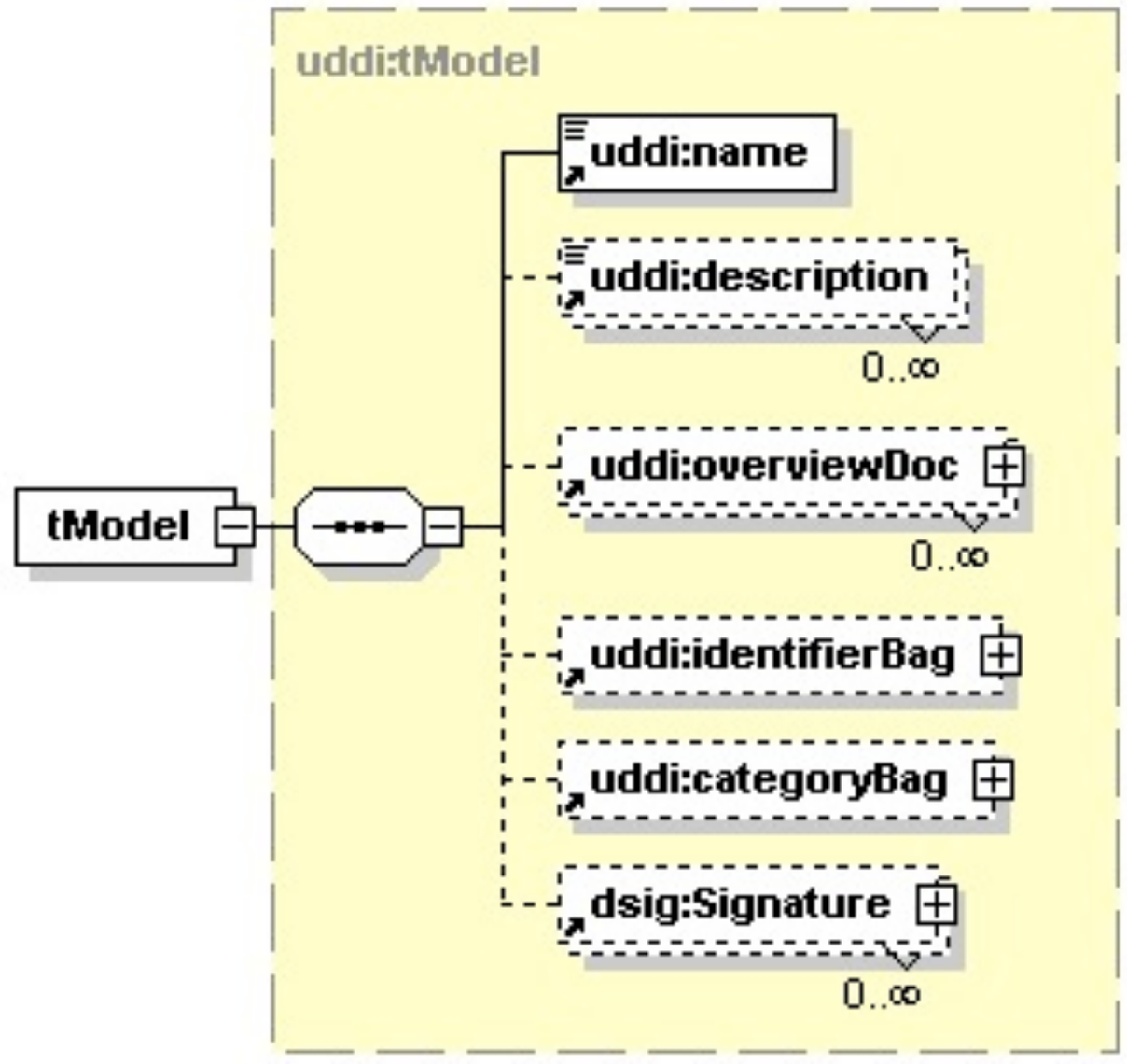


bindingTemplate accessPoint

- Network address of the Web service being described, further classified by *useType* attribute
  - endPoint: accessPoint point to service endpoint network address for invocation
  - bindingTemplate: cross-reference other bindingTemplate
  - hostingRedirector: accessPoint must be determined by querying another UDDI registry
  - wsdlDeployment: accessPoint points to a remotely hosted WSDL document, that already contains binding and endpoint information

tModel

- *name, description*: Textual information about tModel
  - name should be an URI
- *overviewDoc*: Optional repeating element with references to remote descriptions
- *identifierBag, categoryBag, Signature*
- Usage for
  - Technical specifications
  - Identification or categorization of UDDI entities

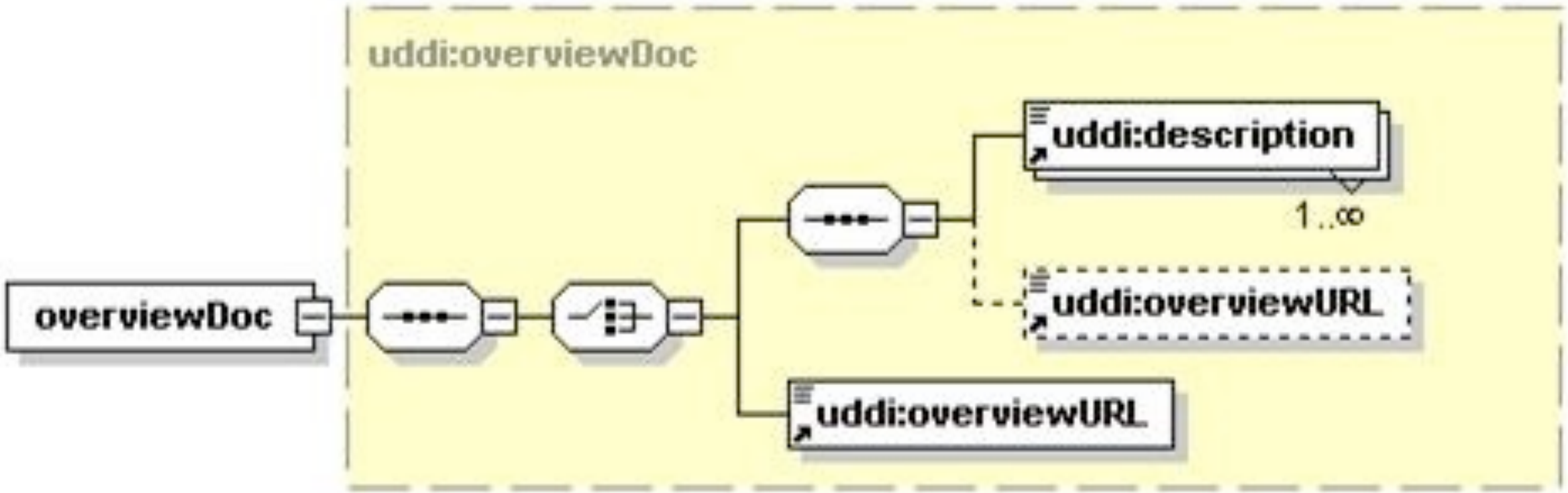


UDDI tModels

- Adding property information to an UDDI entity:
  “Mark a description with information that designates how it behaves, what conventions it follows, and what specifications or standards the service complies with.”
    - *tModel* re-use in *bindingTemplates* can express interoperability
        - Same set of *tModel* elements leads to same technical fingerprint
- Technical documents necessary to a developer using Web services are not stored within the UDDI registry itself
    - tModel contains the addresses where those documents can be found
- Each tModel instance is a keyed entity in UDDI
    - Sources for determining compatibility of Web services
    - Keyed namespace references

overviewDoc

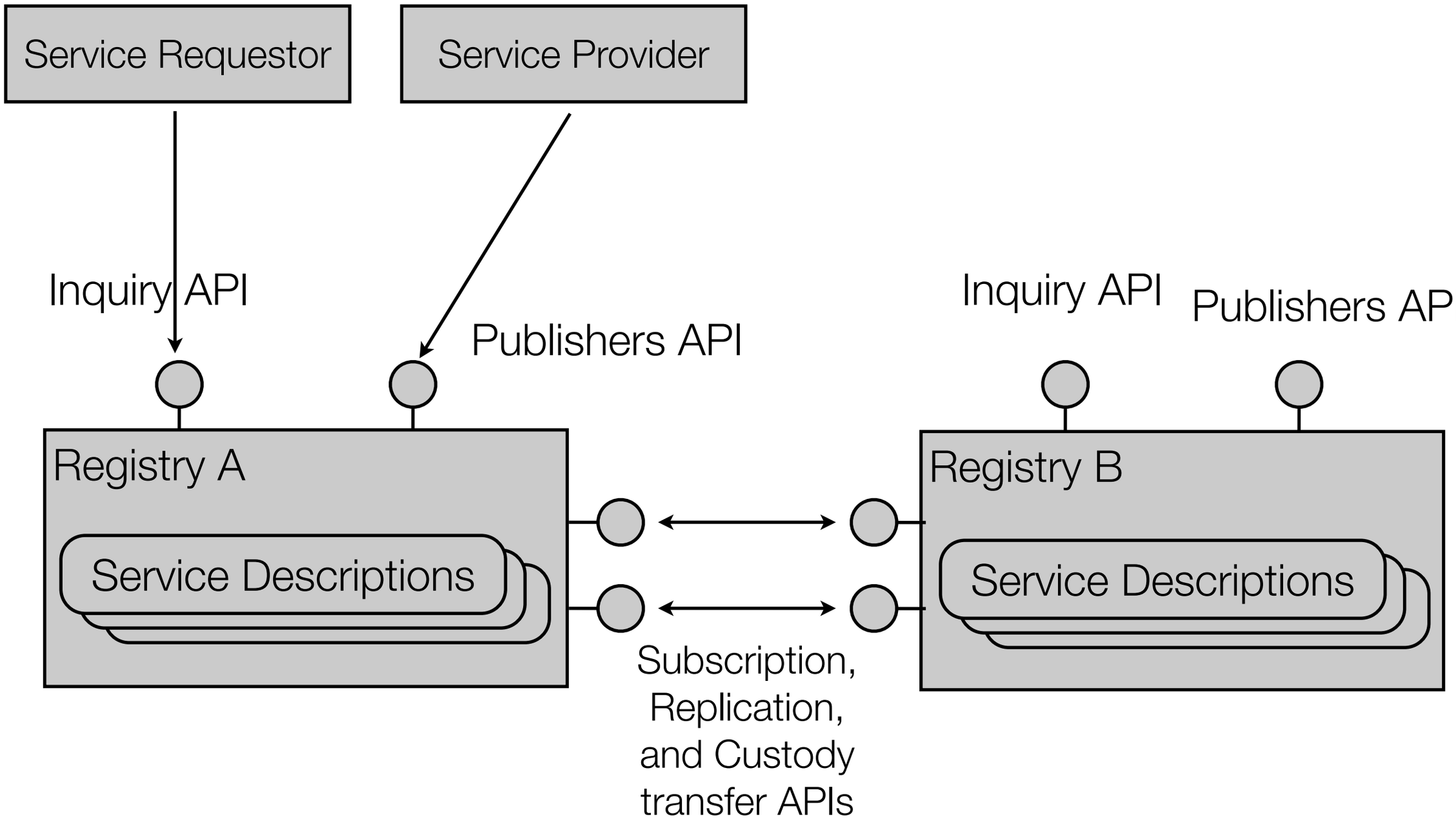
- description: mandatory repeating element, descriptive overview of how the model is to be used
- overviewURL: Long form of usage guide



UDDI APIs

- Different user roles considered by different APIs, each with own access point
  - *UDDI Inquiry API*: Find registry entries that satisfy search criteria, and get detailed information
  - *UDDI Publishers API*: Add, modify, and delete entries; intended for service providers
  - *UDDI Security API*: Allows users to get and discard authentication tokens
  - *UDDI Custody and Ownership Transfer API*: Transfer information ownership between directories and publishers
  - *UDDI Subscription API*: Monitoring of changes by tracking entry creation, deletion, or modification
  - *UDDI Replication API*: Synchronization of registries

UDDI Interactions



```
<find_business generic="1.0" xmlns="urn:uddi-org:api">
<name>Microsoft</name>
</find_business>
```

```
<businessList generic="1.0"
 operator="Microsoft Corporation"
 truncated="false"
 xmlns="urn:uddi-org:api">
 <businessInfos>
  <businessInfo
     businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3">
   <name>Microsoft Corporation</name>
    <description xml:lang="en">
        Empowering people through great software -
        any time, any place and on any device is Microsoft's
        vision. As the worldwide leader in software for personal
        and business computing, we strive to produce innovative
        products and services that meet our customer's
    </description>
    <serviceInfos>
    <serviceInfo
       businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
       serviceKey="611C5867-384E-4FFD-B49C-28F93A7B4F9B">
    <name>Volume Licensing Select Program</name>
    </serviceInfo>
    </serviceInfo>
    </serviceInfos>
  </businessInfo>
 </businessInfos>
</businessList>
```

Inquiry API

- Simple API for searching the UDDI registry
  - Queries are not propagated between registries
  - Specified as Web service interface (http://uddi.org/wsdl/)
- Browsing the registry, e.g. *find_business(), find_service(), find_binding(), find_tModel()*
- Fetch specific entity, e.g. *get_businessDetail()*
- Invoke a service
  - Fetch and cache *bindingTemplate* information, refresh by *get_bindingDetail* when the call fails
- General find qualifiers: *exactMatch, caseSensitiveSort, signaturePresent*, ...

API Call Examples

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <find_business xmlns="urn:uddi-org:api_v2" generic="2.0">
         <categoryBag><keyedReference
            keyValue="51121"
            tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2" />
         </categoryBag>
      </find_business>
   </Body>
</Envelope>
```

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <find_business xmlns="urn:uddi-org:api_v2" generic="2.0">
       <findQualifiers><findQualifier>orAllKeys</findQualifier>
       </findQualifiers>
       <categoryBag>
            <keyedReference keyValue="51121"
                tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2" />
            <keyedReference keyValue="US-OR"
                tModelKey="uuid:4e49a8d6-d5a2-4fc2-93a0-0411d8d19e88" />
         </categoryBag>
      </find_business>
   </Body>
</Envelope>
```

API Specification with tModels

```
<tModel tModelKey="uddi:uddi.org:v3_inquiry">
  <name>uddi-org:inquiry_v3</name>
  <description>UDDI Inquiry API V3.0</description>
  <overviewDoc>
    <overviewURL useType="wsdlInterface">
       http://uddi.org/wsdl/uddi_api_v3_binding.wsdl#UDDI_Inquiry_SoapBinding
    </overviewURL>
  </overviewDoc><overviewDoc>
    <overviewURL useType="text">
         http://uddi.org/pubs/uddi_v3.htm#InqV3
    </overviewURL>
  </overviewDoc><categoryBag>
    <keyedReference keyName="uddi-org:types:wsdl"
      keyValue="wsdlSpec"
      tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyName="uddi-org:types:soap"
      keyValue="soapSpec"
      tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyName="uddi-org:types:xml"
      keyValue="xmlSpec"
      tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyName="uddi-org:types:specification"
      keyValue="specification"
      tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag></tModel>
```

UDDI Types Category System

- Distinguish various types of concepts, described by tModels
- Publishers are encouraged to categorize their tModels by these values
- Possible keyValue attributes for “uddi:uddi.org:categorization:types” tModel references in categoryBags; usable through find_* calls
  - *tModel*: Root of the branch of the category system that is intended for use in categorization of tModels within the UDDI registry
  - *valueSet*: Parent branch for the identifier, namespace, and categorization values
  - *identifier*: Represents a specific set of values used to uniquely identify information
    - Intended to be used in keyedReferences inside of identifierBags
    - Example: Dun & Bradstreet D-U-N-S® Number uniquely identifies companies

UDDI Types Category System

- *namespace*: Represents a scoping constraint or domain for a set of information
  - Does not have a predefined set of values within the domain (like identifier)
  - Acts to avoid collisions, similar to the namespace functionality in XML
- *categorization*: Used for category systems within the UDDI registry
  - Example: tModel for North American Industry Classification System (NAICS)
- *postalAddress*: Used to identify different forms of postal addresses
  - May be used with the *address* element
- *sortOrder*: Defines a collation sequence that can be used during inquiries to control ordering of the results

UDDI Types Category System

- *specification*: Used for tModels that define interactions with a Web service
  - Definition of the set of requests and responses
  - Examples: tModels describing XML, COM or CORBA interaction
- *xmlSpec*: Refinement of the specification tModel type
  - Indicate that the interaction with the Web service is via XML
  - Example: UDDI API tModels
- *soapSpec*: Further refining the xmlSpec tModel type
  - Indicate that the interaction with the Web service is via SOAP
- *wsdlSpec*: For Web service tModels, which are described by WSDL

UDDI Types Category System

- *protocol*: A tModel describing a protocol of any sort.
- *transport*: Specific type of protocol
  - Example: HTTP, FTP, and SMTP tModels
- *wsdlDeployment*: A bindingTemplate categorized as a *wsdlDeployment* contains within its *accessPoint* the endpoint for a WSDL document

```
<bindingTemplate bindingKey="uddi:example.org:catalog">
   <description xml:lang="en">
       Browse catalog Web service
   </description>
   <accessPoint useType="wsdlDeployment">
       http://www.example.org/CatalogWebService/catalog.wsdl
   </accessPoint>
   <categoryBag>
       <keyedReference keyName="uddi-org:types:wsdl"
       keyValue="wsdlDeployment"
       tModelKey="uddi:uddi.org:categorization:types"/>
   </categoryBag>
</bindingTemplate>
```

bindingTemplate for Inquiry API tModel

```
<bindingTemplate bindingKey="…" serviceKey="…">
  <description>UDDI Inquiry API V3</description>
  <accessPoint useType="endpoint">
     http://URL_of_service
  </accessPoint>
  <tModelInstanceDetails>
    <tModelInstanceInfo
      tModelKey="uddi:uddi.org:v3_inquiry">
      <instanceDetails>
        <instanceParms>
          <![CDATA[
            <?xml version="1.0" encoding="utf-8" ?>
              <UDDIinstanceParmsContainer
               xmlns="urn:uddi-org:policy_v3_instanceParms">
                <defaultSortOrder>
                  uddi:uddi.org:sortorder:binarysort
                </defaultSortOrder>
              </UDDIinstanceParmsContainer>
          ]]>
        </instanceParms>
      </instanceDetails>
    </tModelInstanceInfo>
  </tModelInstanceDetails>
</bindingTemplate>
```

tModelInstanceDetails

- *tModelKey* attribute: References a tModel with which the Web service represented by the containing *bindingTemplate* complies
- optional *instanceDetails* element: tModel-specific settings
  - Required to describe a tModel specific component of a service description
  - Support services that require additional technical data support,e.g. via settings or other handshake operations
- *instanceParms:* Optional element of type string
  - Used to locally contain settings or parameters related to the proper use
  - Suggested format is a namespace-qualified XML document

```
<businessService>
  <name>Island Trading Tame Animal Catalog Service</name>
  <description xml:lang="en">
     Search our Tame animals catalog on line
  </description>
  <bindingTemplates>
    <bindingTemplate>
      <accessPoint useType="endpoint">
         https://islandtrading.example/tame/catalog.html
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo
          tModelKey="uddi:uddi.org:ubr:transport:http">
        </tModelInstanceInfo>
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
  <categoryBag>
    <keyedReference
       tModelKey="uddi:uddi.org:categorization:general_keywords"
       keyName="islandtrading.example:categorization:animals"
       keyValue="c"/>
    <keyedReference
       tModelKey="uddi:uddi.org:ubr:categorization:unspsc"
       keyName="UNSPSC: Livestock" keyValue="101015"/>
  </categoryBag>
</businessService>
```

Public UDDI Registries

"IBM and Microsoft have pulled the plug on the Universal Description, Discovery and Integration (UDDI) public registry. The shutdown comes more than five years after an impressive list of partners announced UDDI in September, 2000. ...

The public UDDI registry grew to include 50,000 entities but the **vision of UDDI as a global e-business registry never materialized.** The post-9/11 economic downturn put a damper on the prospects for global e-markets, but there has been slow, steady progress in that direction. The adoption rate for e-business messaging and registry technologies, such as UDDI, is greater in Asia and Europe than in North America. For example, the Australian Government and Standards sponsor BizDex, a national B2B registry.

After a few years, **it became apparent that private registries had become UDDI's biggest success.** Although they will no longer provide public registries, IBM and Microsoft continue to support the UDDI specification and private registries. IBM bundles UDDI 3.0 with WebSphere Application Server and some of its Rational developer tools."

Ken North, http://www.webservicessummit.com/News/UDDI2006.htm