Modeling the UDDI Schema with UML

Dave Carlson
CTO
Ontogenics Corp.
Boulder, Colorado
dcarlson@ontogenics.com
http://XMLModeling.com

Complex XML vocabulary definitions are often easier to comprehend and discuss with others when they are expressed graphically. Although existing tools for editing schemas provide some assistance in this regard (e.g., Extensibility XML Authority) they are generally limited to a strict hierarchical view of the vocabulary structure. More complex structures are often represented in schemas using a combination of containment and link references (including ID/IDREF, simple href attributes, and more flexible extended XLink attributes). These more object-oriented models of schema definition are more easily represented using UML class diagrams [1].

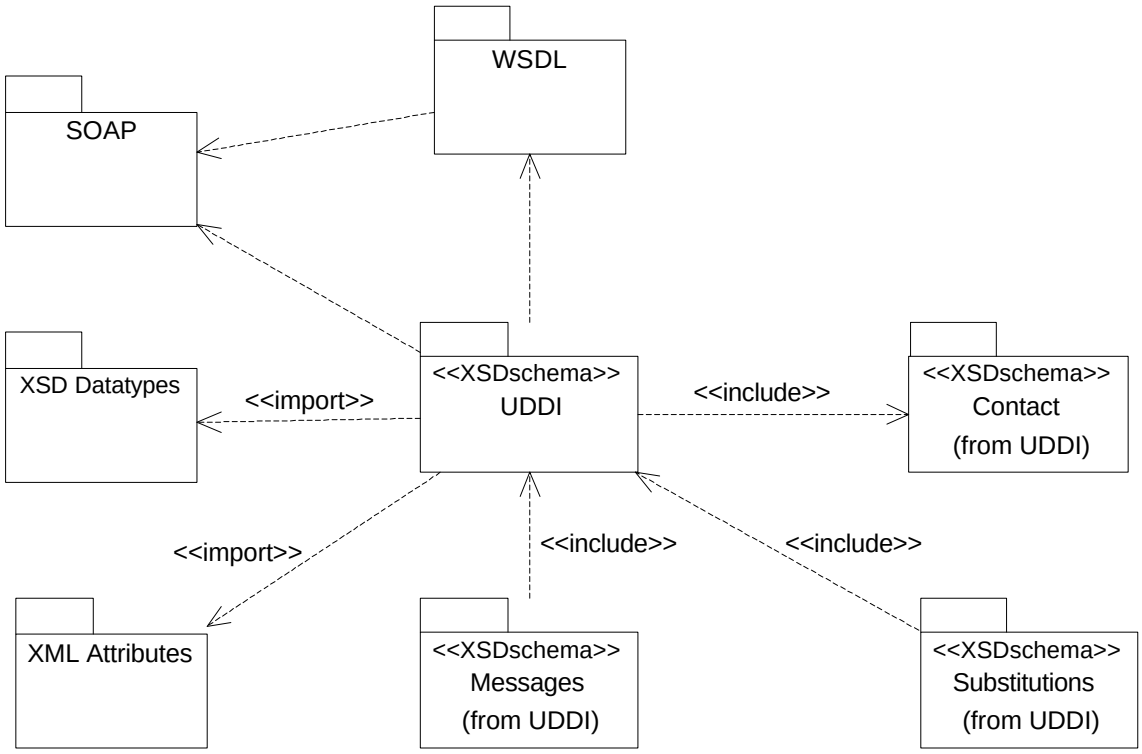
The Unified Modeling Language (UML) is the most recent evolution of graphical notations for object-oriented analysis and design, plus it includes the necessary metamodel that defines the modeling language itself. UML has experienced rapid growth and adoption in the last two years, due in part to its vendor-independent specification as a standard by the Object Management Group (OMG). This UML standard and metamodel are the basis for a new wave of modeling tools by many vendors. For more information and references to these topics, see the Web portal at XMLModeling.com, which was created for the purpose of communicating information about the intersection of XML and UML technologies.

The OMG has also adopted a standard interchange format for serializing models and exchanging them between UML tools, called the XML Metadata Interchange (XMI) specification [2]. (XMI is actually broader in scope than this, but its use with UML is most prevalent.) Many UML modeling tools now support import/export using the XMI format, so it's now possible to get an XML document that contains a complete UML model definition. This capability is the foundation for the remainder of this white paper. Using an XMI file exported from any UML tool, I have written an XSLT transformation that generates an XML Schema document from the UML class model definition.

As an example that demonstrates the benefits of modeling XML vocabularies with UML, I have reverse-engineered a substantial part of the UDDI specification [3]. The current UDDI specification (as of January 2001) includes an XML Schema definition that is based on an old version of the XML Schema draft, well before the out-dated April 7th 2000 draft. In contrast, the UDDI schemas described here are compliant with the XML Schema Candidate Recommendation, dated October 24, 2000. The reverse engineering was accomplished manually, by reading the UDDI specification and creating a UML model in Rational Rose.

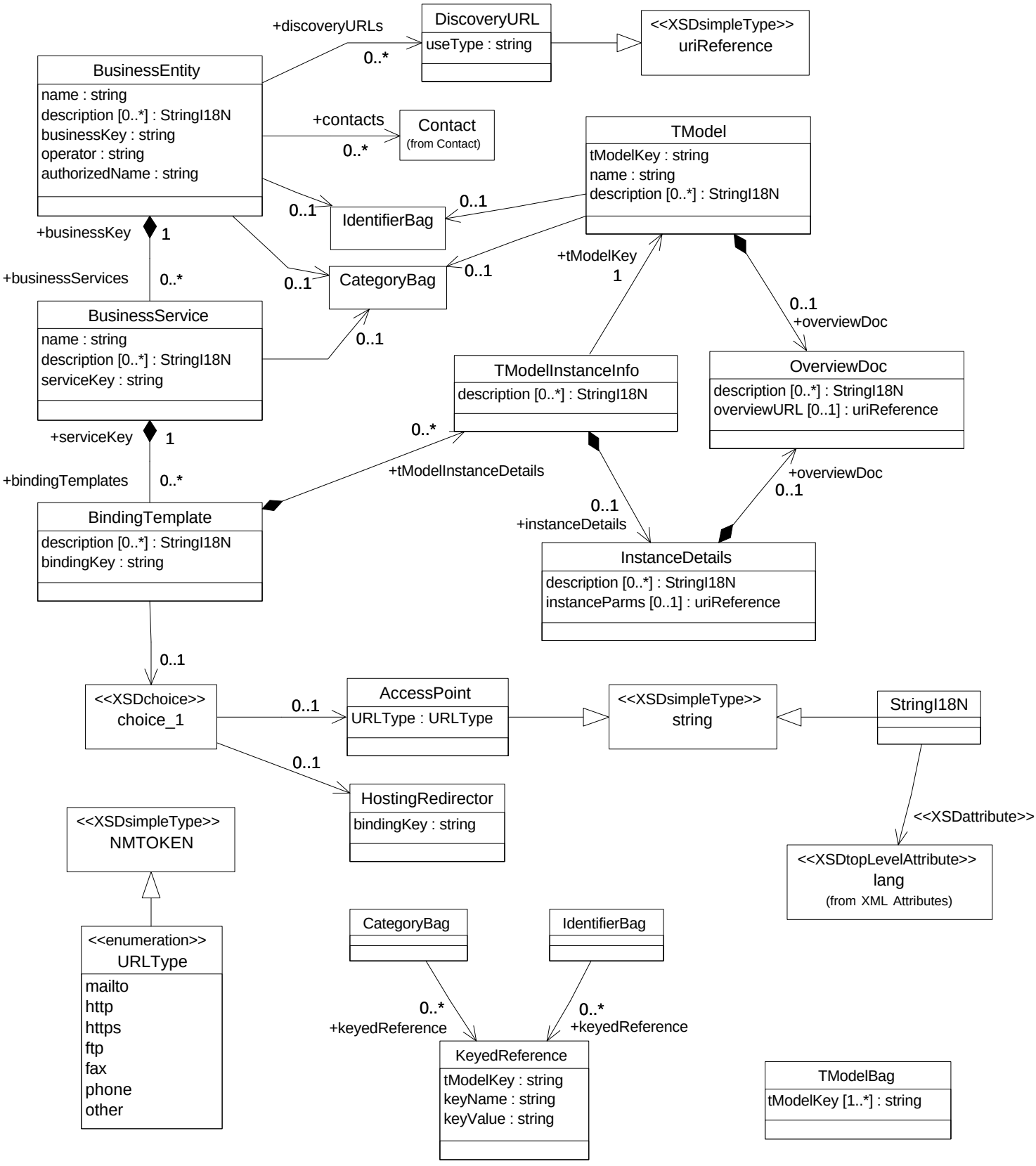
This is work in progress! I spent only a few days studying the UDDI specification and schema while building this model. I then applied a prototype tool that generates XML Schemas from UML, with the goal of regenerating a schema that is semantically equivalent to the one included in the UDDI specification. I used the jUDDI client application [4] to query a business description document from Microsoft's UDDI repository. That XML document is, of course, represented using elements from the UDDI schema. Finally, using the XML Spy 3.5 beta tool, the UDDI instance document was successfully validated using the schema generated from this UML model.

UML enables definition of a model structure using packages; these are illustrated in a diagram using file folder icons, optionally connected with dependency arrows. A high-level view of UDDI and its relationship to other specifications is shown in the following figure. The following examples expand the contents of the three packages named UDDI, Contact, and Messages in the diagram.



UDDI Core Content Model

The following UML class diagram illustrates all of the core elements from the UDDI specification. There are several important points to notice. First, I followed a common software design practice when naming the classes by beginning class names with an upper-case letter; the UDDI schema uses lower-case letters to begin all element names. Several other prominent XML schemas also adopt the more object-oriented naming conventions as used here. This design requires additional UML and schema elements to be declared that map lower-case names to their upper-case complexType definitions. Second, in order to allow a designer control over the XML schema generation process, the model was defined with several UML stereotypes and tagged values (a standard technique that is part of UML). Most of these are not shown in this diagram, but they did affect the generated schema. Although you can generate a schema from this UML model without using any stereotype extensions, it is unlikely to match the structure of a reverse-engineered schema without some fine-tuning.


DiscoveryURL
useType : string
+discoveryURLs
0..*
<<XSDsimpleType>>
uriReference
BusinessEntity
name : string
description [0..*] : StringI18N
businessKey : string
operator : string
authorizedName : string
+contacts
0..*
Contact
(from Contact)
TModel
tModelKey : string
name : string
description [0..*] : StringI18N
0..1
IdentifierBag
0..1
+businessKey
1
+businessServices
0..*
0..1
CategoryBag
0..1
+tModelKey
1
0..1
+overviewDoc
BusinessService
name : string
description [0..*] : StringI18N
serviceKey : string
0..1
TModelInstanceInfo
description [0..*] : StringI18N
OverviewDoc
description [0..*] : StringI18N
overviewURL [0..1] : uriReference
+serviceKey
1
+bindingTemplates
0..*
0..*
+tModelInstanceDetails
+overviewDoc
0..1
BindingTemplate
description [0..*] : StringI18N
bindingKey : string
0..1
+instanceDetails
InstanceDetails
description [0..*] : StringI18N
instanceParms [0..1] : uriReference
0..1
<<XSDchoice>>
choice_1
0..1
AccessPoint
URLType : URLType
<<XSDsimpleType>>
string
StringI18N
0..1
HostingRedirector
bindingKey : string
<<XSDsimpleType>>
NMTOKEN
<<XSDattribute>>
<<XSDtopLevelAttribute>>
lang
(from XML Attributes)
CategoryBag
IdentifierBag
<<enumeration>>
URLType
mailto
http
https
ftp
fax
phone
other
0..*
+keyedReference
0..*
+keyedReference
KeyedReference
tModelKey : string
keyName : string
keyValue : string
TModelBag
tModelKey [1..*] : string

I chose the UDDI schema as the subject of this reverse-engineering exercise in part because of its complexity. If this can be accomplished successfully, then other projects are also likely to be feasible. In addition, if the UML modeling was done as an integral part of the original schema analysis and design, then a number of inconsistencies could be resolved and the schema could be generated using fewer stereotype extensions.

XML Schema definitions are shown for four of the UML model classes. These were generated from the XMI export of this model using an XSLT stylesheet to execute the transformations. They were not manually edited. The schema generator also includes full support for generalization (i.e., inheritance) in the UML model and produces valid XML Schema definitions using both complexType and simpleType derivation. The UML stereotypes and associated tagged values provide control over design issues such as: selection of <choice>, <sequence>, or <all> content model for each complexType definition; specification of the element order within <sequence> (UML attributes are unordered in the model); selection of whether UML attributes and association roles are generated as attributes or elements in the XML Schema; and control over the way that UML association ends are embedded with or without role elements, and as 'ref' or 'type' declarations.

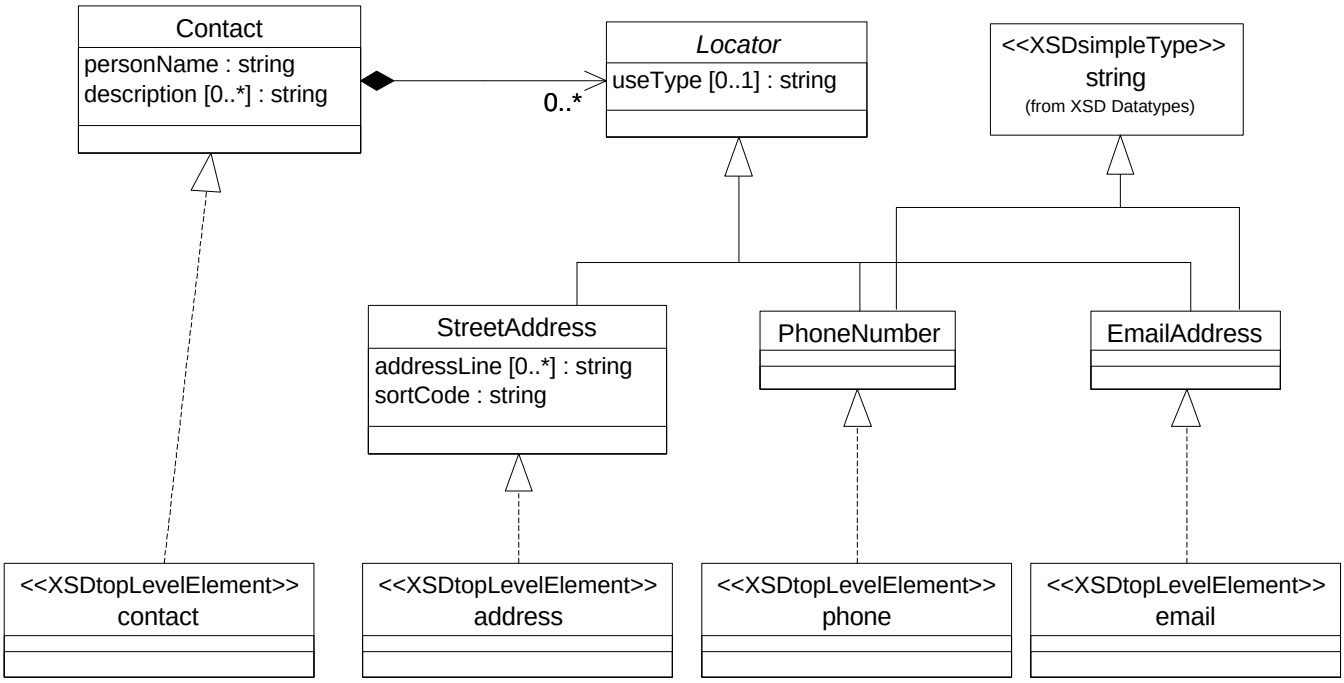
```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: BusinessEntity  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

   <xs:element name="BusinessEntity" type="uddi:BusinessEntity"/>
   <xs:complexType name="BusinessEntity">
      <xs:annotation>
         <xs:documentation>
           Describes an instance of a business or business unit.
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="discoveryURLs" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="uddi:DiscoveryURL"
                              minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="name" type="xsd:string"/>
         <xs:element name="description" type="uddi:StringI18N"
                     minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="contacts" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="uddi:Contact"
                              minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="businessServices" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="uddi:BusinessService"
```

```
                              minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
             </xs:complexType>
          </xs:element>
          <xs:element ref="uddi:IdentifierBag" minOccurs="0" maxOccurs="1"/>
          <xs:element ref="uddi:CategoryBag" minOccurs="0" maxOccurs="1"/>
       </xs:sequence>
       <xs:attributeGroup ref="uddi:BusinessEntity.att"/>
    </xs:complexType>
    <xs:attributeGroup name="BusinessEntity.att">
       <xs:attribute name="businessKey" type="xsd:string" use="required"/>
       <xs:attribute name="operator" type="xsd:string" use="required"/>
       <xs:attribute name="authorizedName" type="xsd:string" use="required"/>
    </xs:attributeGroup>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: BusinessService  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

    <xs:element name="BusinessService" type="uddi:BusinessService"/>

    <xs:complexType name="BusinessService">
       <xs:annotation>
          <xs:documentation>Describes a logical service type in business terms.
          </xs:documentation>
       </xs:annotation>
       <xs:sequence>
          <xs:element name="name" type="xsd:string"/>
          <xs:element name="description" type="uddi:StringI18N"
                      minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="bindingTemplates" minOccurs="0" maxOccurs="1">
             <xs:complexType>
                <xs:sequence>
                   <xs:element ref="uddi:BindingTemplate"
                               minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
             </xs:complexType>
          </xs:element>
          <xs:element ref="uddi:CategoryBag" minOccurs="0" maxOccurs="1"/>
       </xs:sequence>
       <xs:attributeGroup ref="uddi:BusinessService.att"/>
    </xs:complexType>
    <xs:attributeGroup name="BusinessService.att">
       <xs:attribute name="serviceKey" type="xsd:string" use="required"/>
       <xs:attribute name="businessKey" type="xs:string"/>
    </xs:attributeGroup>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: CategoryBag  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

    <xs:element name="CategoryBag" type="uddi:CategoryBag"/>
    <xs:complexType name="CategoryBag">
       <xs:sequence>
          <xs:element name="keyedReference" type="uddi:KeyedReference"
                      minOccurs="0" maxOccurs="unbounded"/>
```

```
      </xs:sequence>
      <xs:attributeGroup ref="uddi:CategoryBag.att"/>
   </xs:complexType>
   <xs:attributeGroup name="CategoryBag.att"/>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: KeyedReference  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

   <xs:element name="KeyedReference" type="uddi:KeyedReference"/>
   <xs:complexType name="KeyedReference">
      <xs:attributeGroup ref="uddi:KeyedReference.att"/>
   </xs:complexType>
   <xs:attributeGroup name="KeyedReference.att">
      <xs:attribute name="tModelKey" type="xsd:string" use="required"/>
      <xs:attribute name="keyName" type="xsd:string" use="required"/>
      <xs:attribute name="keyValue" type="xsd:string" use="required"/>
   </xs:attributeGroup>
```

Contact Package

These elements related to business contact information are part of the core UDDI schema. However, I have created a separate package in this UML model for three reasons. First, it's important to promote the separation and reuse of common components. This is a fundamental goal in object-oriented analysis and design, and contact or party information is a classic example of common elements required in most e-business applications. Second, because this is primarily an educational exercise, I wanted to create a second UML package that would be included by the schema transformation tool. Third, and also part of the educational goal, this package includes an experimental mapping for multiple inheritance in UML models to XML Schema. Schema does not support multiple inheritance.



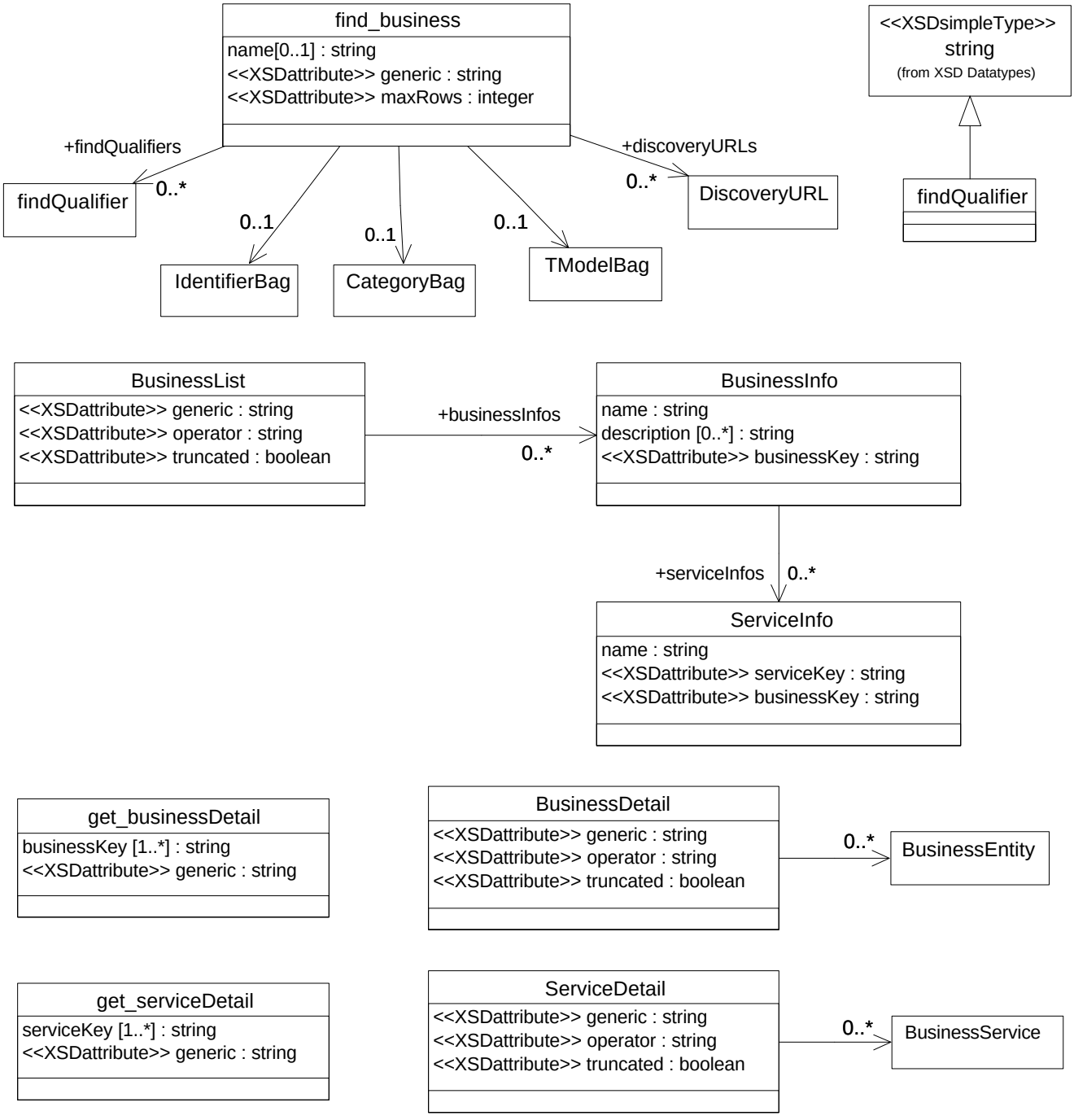
The Locator class is an abstract class in UML (denoted by its name in italics in the diagram), so it is generated to a complexType with an abstract='true' attribute. StreetAddress uses single inheritance, so it may use complexType extension to inherit the attributes of Locator in the schema. PhoneNumber inherits from both Locator (a complexType) and string (a simpleType drawn from the XSD Datatypes package). In this mapping, a simpleType superclass takes precedence and generates the corresponding simpleContent child in PhoneNumber. The Locator attributes are copied down and duplicated within the content model for PhoneNumber. Finally, all three subclasses include a substitutionGroup attribute that allows them to be substituted for Locator within the content of a Contact element.

As mentioned previously, this model uses upper-case names for the model elements, whereas the UDDI schema uses lower-case names. Thus, this model includes four additional classes stereotyped as << XSDtopLevelElement>> that create lower-case elements in the generated schema (not shown in the code examples). These classes are connected with a UML "realization" relationship in this mapping. This contact model creates a schema that is slightly more lenient than the original UDDI schema, but it successfully validates current UDDI documents and I believe that it more accurately reflects the intended conceptual model of UDDI requirements (with the exception of upper/lower-case issues).

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: Locator  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

   <xs:element name="Locator" type="uddi:Locator"/>
   <xs:complexType name="Locator" abstract="true">
      <xs:attributeGroup ref="uddi:Locator.att"/>
   </xs:complexType>
   <xs:attributeGroup name="Locator.att">
      <xs:attribute name="useType" type="xsd:string"/>
   </xs:attributeGroup>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: StreetAddress  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

   <xs:element name="StreetAddress" type="uddi:StreetAddress"
               substitutionGroup="uddi:Locator"/>
   <xs:complexType name="StreetAddress">
      <xs:annotation>
         <xs:documentation>
           A printable, free-form address. Typed by convention. Sort not used.
         </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
         <xs:extension base="uddi:Locator">
            <xs:sequence>
               <xs:element name="addressLine" type="xsd:string"
                           minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attributeGroup ref="uddi:StreetAddress.att"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:attributeGroup name="StreetAddress.att">
      <xs:attribute name="sortCode" type="xsd:string" use="required"/>
   </xs:attributeGroup>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: PhoneNumber  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

   <xs:element name="PhoneNumber" type="uddi:PhoneNumber"
               substitutionGroup="uddi:Locator"/>
   <xs:complexType name="PhoneNumber">
      <xs:simpleContent>
         <xs:extension base="xsd:string">
            <xs:attributeGroup ref="uddi:Locator.att"/>
            <xs:attributeGroup ref="uddi:PhoneNumber.att"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:attributeGroup name="PhoneNumber.att"/>
```

UDDI Message Model

The UDDI specification is defined by a single schema and that schema also includes elements that are the basis for SOAP messages. I split these definitions into two packages in the UML model and both are generated into a single schema document. The following diagram shows a few classes that represent a small subset of the UDDI message structures.



The remaining examples focus on two of these UDDI messages: `get_businessDetail` and `BusinessDetail`. These messages represent a request—response pair in the UDDI protocol and both are wrapped in SOAP message envelopes when exchanged by applications. Notice that the `BusinessDetail` includes an association to zero or more `BusinessEntity` elements, which reference the definition listed in the previous section. Notice also that an extra top-level element was declared for `businessDetail` (not shown in the diagram) that maps to the upper-case complexType named `BusinessDetail` in the schema.

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: get_businessDetail  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

   <xs:element name="get_businessDetail" type="msg:get_businessDetail"/>
   <xs:complexType name="get_businessDetail">
      <xs:sequence>
         <xs:element name="businessKey" type="xsd:string"
                     minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attributeGroup ref="msg:get_businessDetail.att"/>
   </xs:complexType>
   <xs:attributeGroup name="get_businessDetail.att">
      <xs:attribute name="generic" type="xsd:string" use="required"/>
   </xs:attributeGroup>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: BusinessDetail  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

   <xs:element name="BusinessDetail" type="msg:BusinessDetail"/>
   <xs:complexType name="BusinessDetail">
      <xs:sequence>
         <xs:element ref="uddi:BusinessEntity"
                     minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attributeGroup ref="msg:BusinessDetail.att"/>
   </xs:complexType>
   <xs:attributeGroup name="BusinessDetail.att">
      <xs:attribute name="generic" type="xsd:string" use="required"/>
      <xs:attribute name="operator" type="xsd:string" use="required"/>
      <xs:attribute name="truncated" type="xsd:boolean" use="required"/>
   </xs:attributeGroup>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- CLASS: <<XSDtopLevelElement>> businessDetail  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

   <xs:element name="businessDetail" type="msg:BusinessDetail"
               substitutionGroup="msg:BusinessDetail"/>
```

Sample UDDI <businessDetail> Document

When a <get_businessDetail> message is sent to Microsoft's UDDI repository (wrapped in a SOAP message envelope), then the following <businessDetail> message is returned. I've removed the SOAP envelope elements that wrapped this response message and modified the default namespace declaration to assign my test namespace, but otherwise the message is exactly as returned.

```
<businessDetail generic="1.0" operator="Microsoft Corporation" truncated="false"
    xmlns="http://www.xmlmodeling.com/schemas/uddi"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
    xsi:schemaLocation="http://www.xmlmodeling.com/schemas/uddi
                        uddi.xsd">
    <!-- Dave Carlson:
         Modified original default namespace from xmlns="urn:uddi-org:api"
    -->

  <businessEntity authorizedName="Martin Kohlleppel"
                  businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
                  operator="Microsoft Corporation">
    <discoveryURLs>
      <discoveryURL useType="businessEntity">
          http://uddi.microsoft.com/discovery?businessKey=0076B468-EB27-42E5-AC09-
9955CFF462A3</discoveryURL>
    </discoveryURLs>
    <name>Microsoft Corporation</name>
    <description xml:lang="en">Software Vendor</description>
    <contacts>
      <contact>
        <personName>Martin Kohlleppel</personName>
        <email>martink@microsoft.com</email>
      </contact>
    </contacts>
    <businessServices>
      <businessService businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
                       serviceKey="CC02E21A-2B4E-4F1A-A2C0-5C5952D67231">
        <name>UDDI Web Services</name>
        <description xml:lang="en">UDDI Registries and Test Sites</description>
        <bindingTemplates>
          <bindingTemplate bindingKey="2303E786-68C0-426E-A646-8E366B43F1FD"
                           serviceKey="CC02E21A-2B4E-4F1A-A2C0-5C5952D67231">
            <description xml:lang="en">
                Microsoft Production UDDI Registry web interface
            </description>
            <accessPoint URLType="http">http://uddi.microsoft.com</accessPoint>
            <tModelInstanceDetails>
              <tModelInstanceInfo
                     tModelKey="uuid:4CD7E4BC-648B-426D-9936-443EAAC8AE23">
                <description xml:lang="en">UDDI Inquiry Service Type</description>
                <instanceDetails>
                  <description xml:lang="en">SOAP URL for inquire</description>
                  <instanceParms>http://uddi.microsoft.com/inquire</instanceParms>
                </instanceDetails>
              </tModelInstanceInfo>
```

```
              <tModelInstanceInfo
                     tModelKey="uuid:64C756D1-3374-4E00-AE83-EE12E38FAE63">
                <description xml:lang="en">UDDI Inquiry Publication Type
                </description>
                <instanceDetails>
                  <description xml:lang="en">SOAP URL for publication</description>
                  <instanceParms>https://uddi.microsoft.com/publish</instanceParms>
                </instanceDetails>
              </tModelInstanceInfo>
            </tModelInstanceDetails>
          </bindingTemplate>

 . . .

      </businessService>
    </businessServices>
    <identifierBag>
      <keyedReference keyName="D-U-N-S" keyValue="08-146-6849"
                      tModelKey="uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823"/>
    </identifierBag>
    <categoryBag>
      <keyedReference keyName="NAICS: Software Publisher" keyValue="51121"
                      tModelKey="uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2"/>
    </categoryBag>
  </businessEntity>
</businessDetail>
```

References

1. For a quick, very accessible introduction to UML and its graphical notation, see: Martin Fowler, *UML Distilled*, second edition, Addison-Wesley.
2. Object Management Group. *XML Metadata Interchange (XMI)*, version 1.1. See ftp://ftp.omg.org/pub/docs/ad/99-10-02.pdf
3. For current UDDI specifications, see http://www.uddi.org
4. jUDDI is an open-source Java library for UDDI, see http://www.juddi.org
5. A forthcoming book describes processes and design techniques for generating XML DTDs and Schemas from UML models. See: David Carlson, *Modeling XML Applications with UML*, Addison-Wesley, 2001 (to be published in April 2001).
6. A Web portal has been created at http://XMLModeling.com to aggregate newsfeeds and resource references related to modeling XML vocabularies, especially using UML. This site will also contain examples from the book, plus case study examples of modeling XML vocabularies.