An Example of Ignoring Referential Integrity: The LDAP Standard

Michael Blaha (blaha@computer.org)
Modelsoft Consulting Corp
www.modelsoftcorp.com

Abstract

The LDAP standard illustrates the consequences of ignoring referential integrity when building applications.

1. Introduction

In an earlier article on this website (*Referential Integrity Is Important For Databases*, 2005) we explained how referential integrity applies to modeling, programming, and databases. ***Referential integrity*** is a constraint that ensures that references between data are valid and intact. Relational DBMSs have intrinsic support for referential integrity [3], though it often goes unused [2]. Some OO DBMSs also have support as ObjectStore illustrates [1].

Our earlier article alluded to the LDAP standard as an example that demonstrates the peril of ignoring referential integrity and we will now elaborate.

2. The LDAP Standard

The ***Lightweight Directory Access Protocol*** (***LDAP***) is a standard approach for providing directory services. A ***directory*** is a set of objects that are organized into a hierarchy. Directory objects can be of many types, including people, organizations, departments, printers, and documents. Objects in the hierarchy can be accessed through a series of names starting from the root. LDAP has two primary purposes: user authentication and sharing of basic data across applications.

LDAP was originally implemented with files, but many of the current products use databases. The LDAP schema is by intent a meta-schema that stores both a model and the model's data.

3. An LDAP Reverse Engineering Case Study

We first encountered the LDAP standard while performing reverse engineering to better understand a software product. Starting from the product's relational database, we constructed a UML class model. The available inputs were:

- Schema: tables, attributes, data types, nullability, and primary keys.
- Sample data.
- A book explaining LDAP concepts.

4. Reverse Engineering LDAP Tables

We started by typing the schema into a modeling tool. There were eleven tables in all.

Five of the tables had few records and seemed conceptually unimportant, so we ignored them. We were left with six tables with significant application content leading to the metamodel in Figure 1. The annotation 'pk' indicates a primary key. All fields are optional except for those with attribute multiplicity of '[1..1]' (not-null fields).

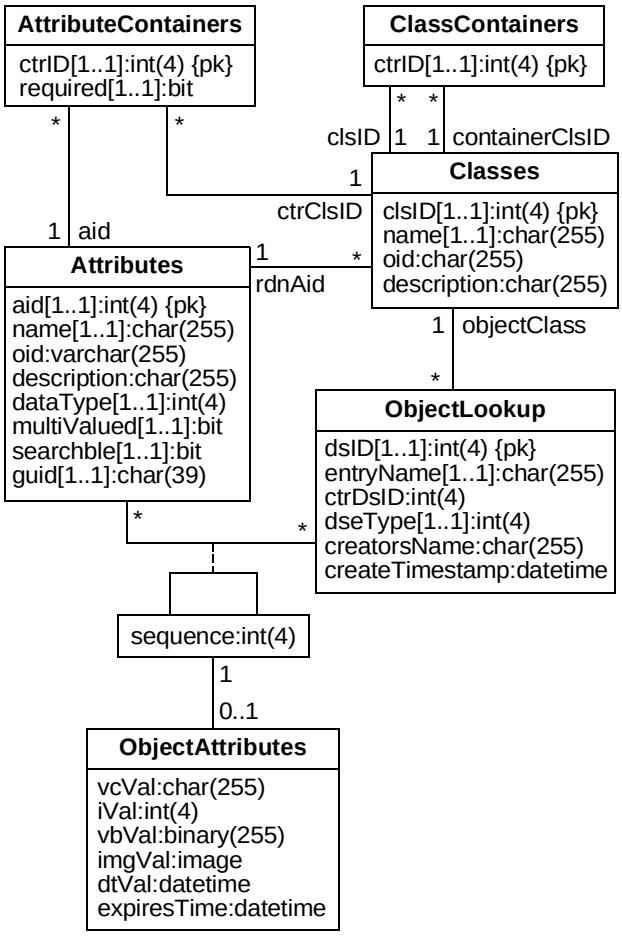

Figure 1 UML class metamodel from reverse engineering

The names are sufficiently clear that most of the model is self evident. *ObjectLookup* stores individual objects, each of which instantiates a class. The combination of an object and an attribute for its class leads to a series of values (*ObjectAttributes*). The containers are used for grouping attributes within classes as well as nesting classes.

We can see that *ObjectAttributes* has parallel data types (*vcVal*, *iVal*, *vbVal*, *imgVal*, and *dtVal*). Each record fills in the one field corresponding to its attribute's data type. By inspecting the data, we found that *ObjectAttributes.iVal* mixes integer values with object references. Therefore the metamodel cannot enforce referential integrity and has the potential for dangling references.

Our concern about dangling references is not an academic curiosity. During the reverse engineering, we were learning about LDAP for the first time. For context, we studied a book about the standard. One of the chapters covered LDAP exception handling for bad references. The entire chapter could have been avoided if the standard had paid attention to referential integrity.

Note that our criticism pertains to the standard and not to the product that we studied. The product faithfully implements the standard, warts and all.

5. Reverse Engineering LDAP Data

Next, we looked at the data that is stored in the LDAP tables and built the model in Figure 2. The boxes denote classes and the arrows show parent-child relationships.

The database has 34 records for the *Classes* table of Figure 1. Figure 2 is a subset of the full model and shows 25 of these classes. Figure 2 does not show attributes for the classes which came from the *Attributes* table. The relationships in *ClassContainers* let us determine the parent-child couplings as the arrows indicate.

Consider the *referral* class of Figure 2. This class has three parents—*top*, *organization*, and *organizationalUnit*. Some referral objects have an *ObjectAttributes.iVal* field that refer to a *top* object. Others have an *iVal* field that refer to *organization* objects. The remainder have an *iVal* field that refer to *organizationalUnit* objects. LDAP forces data into a hierarchy—each object has at most one parent. The model for the data is a network as the objects of a child class may belong to various parent classes.

The LDAP hierarchy cannot properly represent relationships across the levels. Thus many-to-many relationships are assigned a participating class as a parent and the other related class is unenforced. For example, a *meetingParticipant* is a child of *meeting*. Presumably, the *meetingParticipant* also relates to *rtPerson*, but the LDAP data omits this. Once again referential integrity is not enforced with the potential for corrupt references.

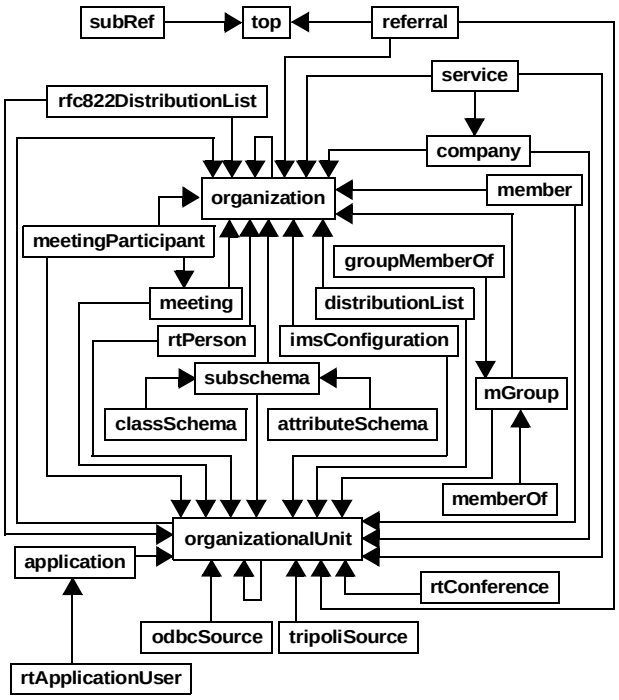

Figure 2 UML class model after reverse engineering

6. Conclusion

The LDAP standard has flawed referential integrity in two ways:

- **Metamodel**. Object references are overloaded onto integer values. As a result, it is not possible to enforce object references.
- **Model**. LDAP forces data into a strict hierarchy. Such an architecture breaks down for many-to-many relationships which intrinsically have two parents. LDAP enforces one of the parents and is lax with the other.

These flaws are part of the LDAP architecture and occur regardless of the delivery platform (files, relational databases, object databases), and regardless of the implementing vendor (as long as the vendor follows the standard).

The referential integrity flaws are ironic for our example. LDAP degrades references even though the underlying relational database can readily support them.

One might ask why the LDAP standard omits support for referential integrity. Might the lack of referential integrity be a deliberate omission in return for the simplicity of a hierarchy? Such a trade-off might be the case, but we suspect not. Most likely the initial LDAP designers did not think beyond the confines of a file. As the standard grew in popularity and was implemented on databases, the architecture was trapped by its origins.

7. References

References [4] and [5] have general LDAP information that we found helpful.

[1] Michael Blaha and William Premerlani. *Object-Oriented Modeling and Design for Database Applications*. Upper Saddle River, New Jersey: Prentice Hall, 1998.

[2] Michael Blaha. A retrospective on industrial database reverse engineering projects—part 2. *Eighth Working Conference on Reverse Engineering*, October 2001, Stuttgart, Germany, 147–153.

[3] Jim Melton and Alan R. Simon. *Understanding the New SQL: A Complete Guide*. San Francisco, California: Morgan Kaufmann, 1993.

[4] http://www.zytrax.com/books/ldap

[5] Rob Weltman and Tony Dahbura. *LDAP Programming with Java*. Reading Massachusetts: Addison-Wesley, 2000.