Data Access Object Pattern

Max Berger

October 30, 2005

Abstract

The data access object (DAO) pattern tries to decouple the access to data from its underlying storage. Persisting data currently relies heavily on the type of database used: Relational database, object-oriented databases, flat files. It would be preferable to chose the type of database used during the deployment phase instead of the design phase. By using data access objects the data is decoupled from its representation, thus allowing to chose different data sources if necessary

1 Introduction

Data access objects provide the portability for applications from one data source to another data source. Many modern applications require a persistent database for their objects. There are currently several common types of databases: Flat files, object-oriented databases and relational databases, with relational databases being the most widely used. Unfortunately these types of databases are accessed in a very different way. Even databases of the same type, such as relational databases behave very similar but not exactly identical. By using data access objects instead of accessing the data source directly, the type and implementation of the actual data source is decoupled from its usage. This allows moving from one data source to a different data source without having to change the business logic.

2 The Data Access Object Pattern

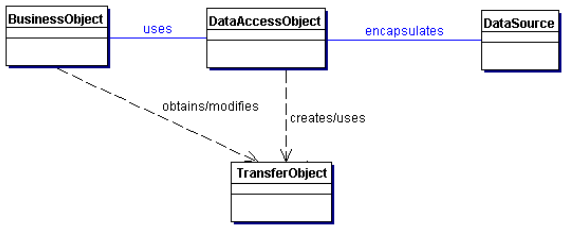

Figure 1: Class diagram representing the relationships for the DAO pattern

The data access object pattern decouples the data from its abstraction by identifying 4 participants: The business object, the data

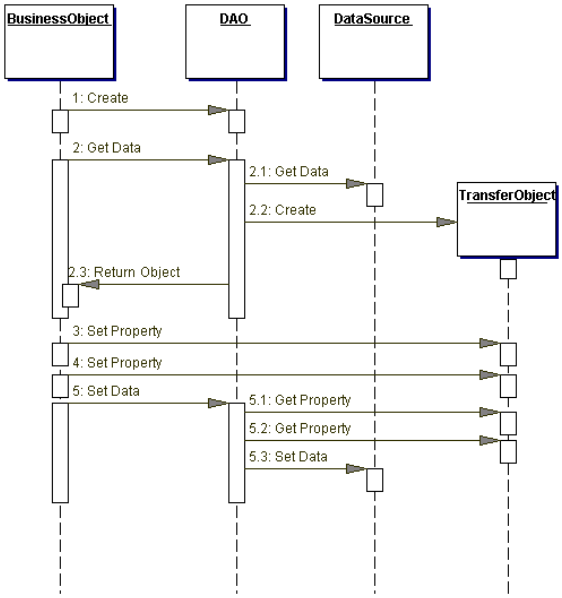

Figure 2: Sequence diagram that shows the interaction between the various participants in this pattern

access object, the data source and a transfer object. Figure 1 shows a class diagram of these participants and their relationships. Figure 2 shows an example interaction between these classes.

The business object represents the class with the business logic. Before the use of DAO this is the class that had all of the responsibilities in it. Now this class is responsible to know what and how to modify the content, and not how to store it.

The data access object hides the actual data source. Instead of talking to the data source directly, the business object has to go through the data access object. Therefore the data access object can be easily be replaced with an object for a different data source.

The data source is the actual data source. In most cases this is some kind of relational database accessed via SQL. It may also be a flat file or an object-oriented database, whatever is available on the deployment platform.

The transfer object is used to transfer the actual data contents from the data access object to the business object and vice versa. It represents the data stored in the database. It is not directly connected to the data source. Any changes to the transfer object must be committed to the data access object before they can become permanent.

2.1 DAO creation strategy

The high flexibility of the DAO pattern comes from the use of the abstract factory and the factory method patterns. Figure 3 shows a possible class diagram for DAO factories.

The DAOFactory class is the base class. It is abstract and provides methods to create DAOs for different objects stored in the database.

The RdbDAOFactory is an implementation of the abstract factory. It has methods to create concrete data access objects, in this case for relational databases.

The RbdDAO1 and RdbDAO2 classes are concrete instances to access a particular piece of information in the database. This example supports what would be two tables in a relational database, so there are two different access objects for the two different pieces of information.

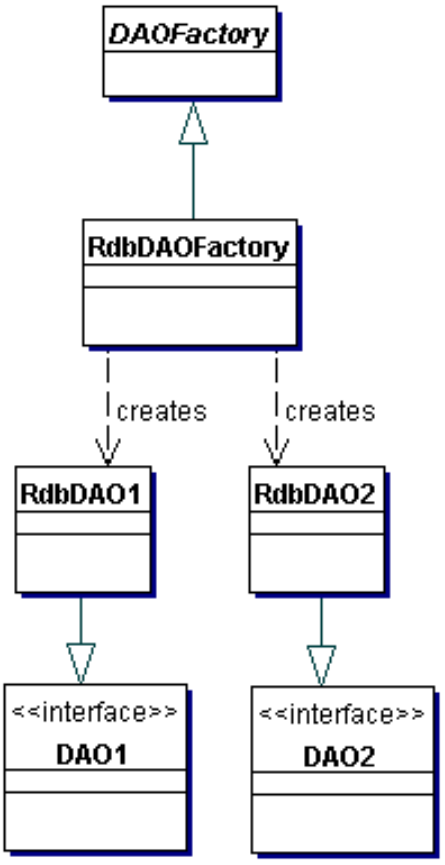

Figure 3: Factory for Data Access Object strategy using Factory Method

The DAO1 and DAO2 interfaces represent this particular piece of information. They provide methods to create, retrieve, update, and delete information (CRUD).

Figure 4 shows a more extensive example with different data sources for relational databases, XML files and object oriented databases. The knowledge about creating a data access object is stored in the concrete factory classes.

Figure 5 shows a possible interaction between these classes for creating a data access object for two different pieces of information stored in a relational database.

3 Application of the DAO in the Metadata Information Database Storage (MIDAS)

The Metadata Information Database Storage (MIDAS) hold file metadata information. It is a module in Sorcers Integrated Local Enhanced New Users Storage (SILENUS). Currently MIDAS is implemented by storing its information in a relational database accessed via the Java Database Connectivity (JDBC). All the paths and scripts are written for the McKoi embedded relational database.

The classes in the current MIDAS will have to be redesigned to work with the new data access object pattern.

The actual data source will be the McKoi relational database. Since the McKoi relational database is accessed via JDBC all common JDBC functionality should be removed any put into a JDBC data access object. Only McKoi related functionality should be in the McKoi database class.

There are two separate pieces of information that have to be kept for the MIDAS. The first one is the information about the files. Files are identified with a UUID. The information they contain is a map of attribute names and values. The second information is

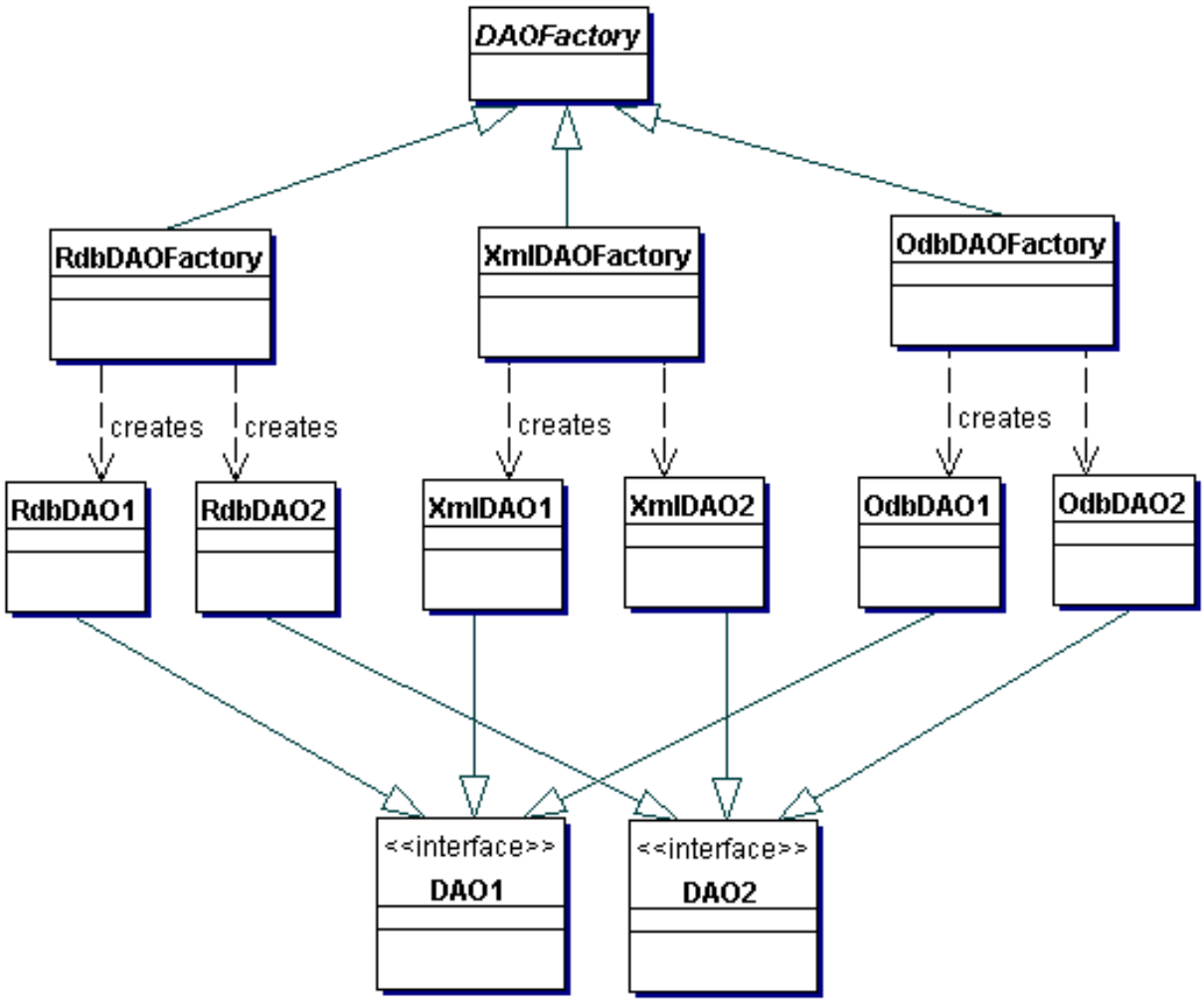

Figure 4: Factory for Data Access Object strategy using Abstract Factory

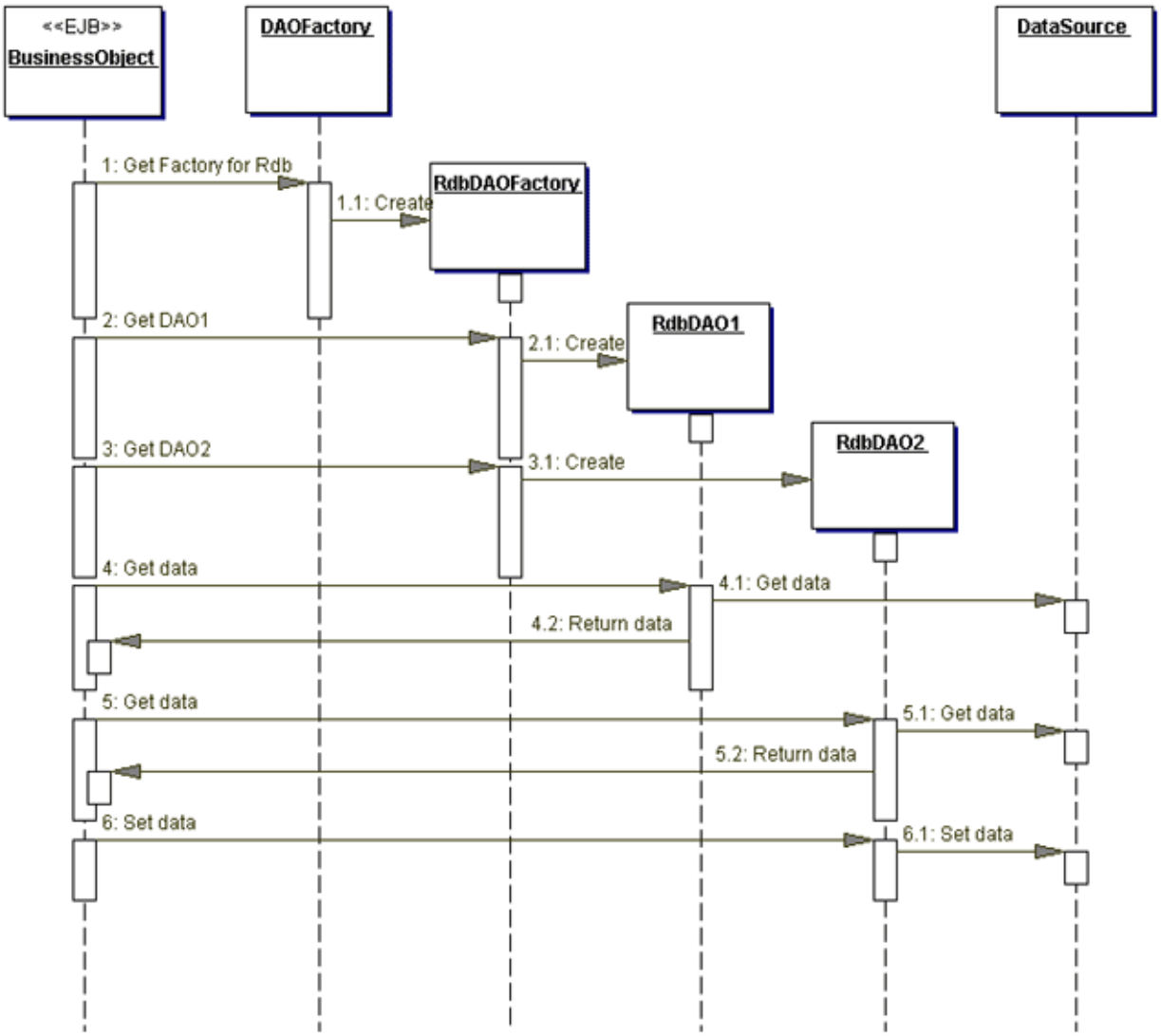

Figure 5: Factory for Data Access Objects using Abstract Factory sequence diagram

attributes for the MIDAS itself. This keeps information such as local and global timestamps which are used for synchronization.

Figure 6 shows a class diagram for the new DAO implementation of the MIDAS. The actual data access has now been split in several parts:

FileAttributesDao is the data access object for the file attributes. It is realized by a generic relational JdbcFileAttributesDao and the more specialized McKoiFileAttributesDao. It provides CRUD functionality for file objects.

MidasAttributesDao is the data access object for the MIDAS attributes. It is also realized by a generic JdbcMidasAttributesDao and a more specialized McKoiMidasAttributesDao class. It provides CRUD functionality to MIDAS attributes.

A generic DaoFactory contains the methods to create both of these access objects. There are implementations for both generic JDBC and McKoi.

4 Extensions to DAO

Several Extensions have been made to the original DAO pattern. There have been several attempts to make the DAO more generic by adding a generic DAO that has support for the 4 basic CRUD functions: Create, Retrieve, Update and Delete. Unfortunately all these solutions sacrifice ease use of for reusability. If the DAOs will have to be used with many different data sources and tables this sacrifice may be worth it. In the given example (MIDAS) there are two data sources. Both are already very generic. Providing a more generic interface will not help in this case but make the use of the data sources more complicated.

5 Conclusions

The DAO pattern can be used to decouple the use of data from its actual storage. It provides support for many types of data sources if used in conjunction with the factory pattern.

The use of the data access object may also encourage the use of additional functionality with proxies, such as caching. This may improve or decrease performance, depending on the use of the data.

The MIDAS example showed that it is very easy to apply the DAO pattern. The two things that had to be done were: Identify the actual data and move the database specific functionality in other classes. These two simple steps give the MIDAS a large boost in flexibility.

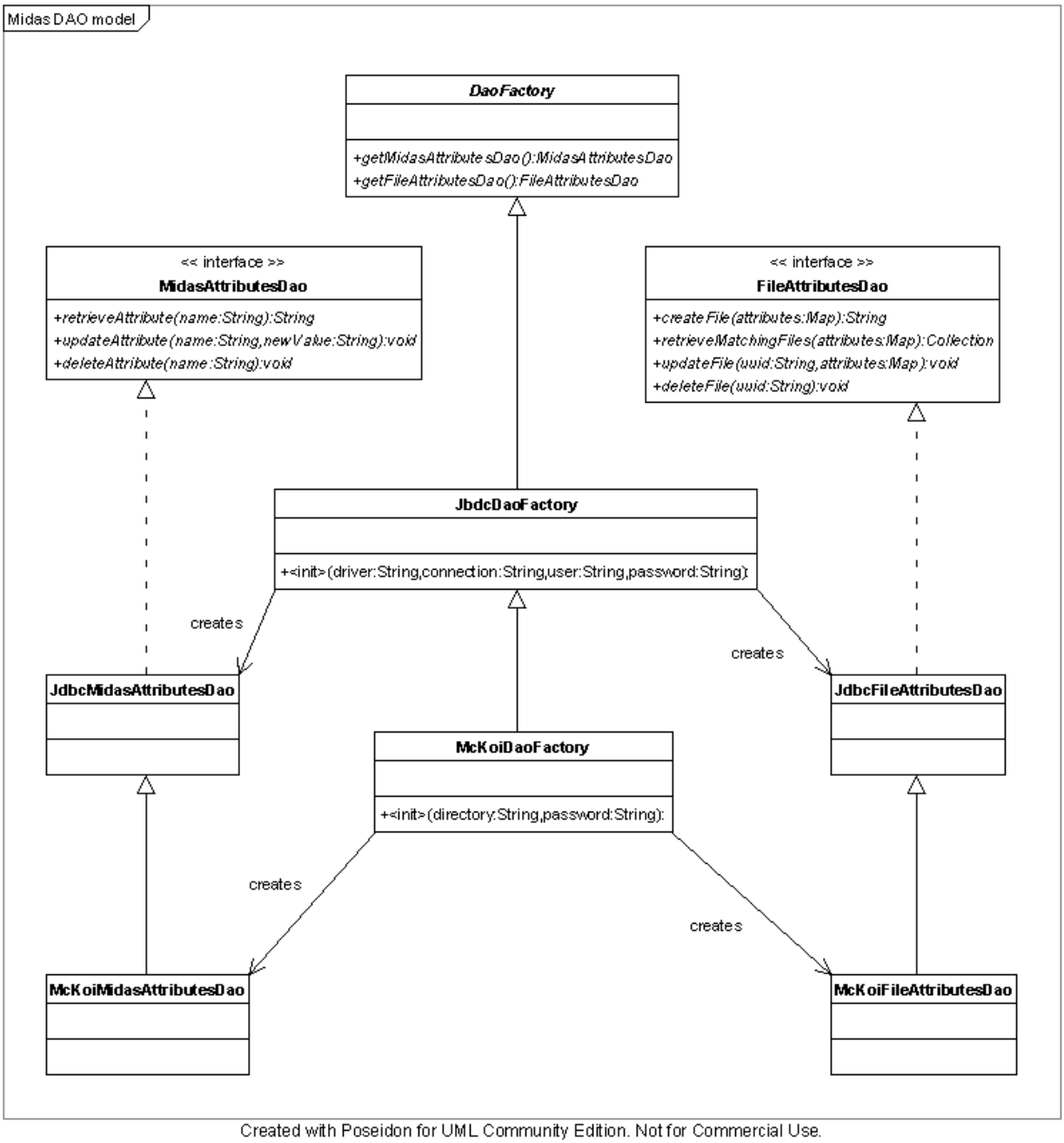

Figure 6: Class diagram for a DAO implementation in Midas