Abstract—Currently existing data access object (DAO) patterns have several limitations. First, the interface of the patterns and business objects is tightly-coupled, which affects seriously the dynamic extensibility of software systems. Second, the patterns have duplicated implementation codes, which add to difficulties of system maintenance. To solve these problems, a new DAO pattern with stronger independency and dynamic extensibility is proposed in this paper. An example is given to illustrate the using process of the new DAO pattern. The greatest advantages of the new DAO pattern are as follows. If any business object is needed to add to the system, we do not have to modify any codes of the class DAO Factory. All we need to do is to modify the mapping file. Furthermore, because we have only one DAO implementation class to accomplish all the data access to business objects, if some SQL statements are needed to be modified, all we need to do is to modify the DAO implementation class but not need to modify any business objects.

Index Terms—Data access object (DAO), data binding, meta model, pattern.

1. Limitations and Challenges

Data access object (DAO) patterns provide an abstract between business logic layer and persistent storage layer through encapsulating all the access to data resources and wrapping up the implementation details of data resources. When the bottom data resources change, the interfaces that DAO patterns provide remain unchanged, which provide convenient migration between different software products, between different types of data storage and between different types of data resources. A generic DAO pattern is shown in Fig. 1.

However, generic DAO patterns have several limitations as we can see from Fig. 1.

(1) Although generic DAO patterns are independent of implementation details of data resources, they are dependent of business objects. Each time we add a new business object to our software system, we have to modify the interface DAOFactory. For example, to add a new business object, named Account, to our software system, we must first create a new interface, named AccountDAO, and then add a new method, named getAccountDAO, to the interface DAOFactory. In a word, the interface DAOFactory and business objects are tightly-coupled, which affects seriously the dynamic extensibility of software systems.

(2) Generic DAO patterns have duplicated implementation codes, which add to difficulties of system maintenance. As Fig. 1 shows, each business object has a corresponding DAO implementation class. For example, business objects Customer and Order have their corresponding DAO implementation classes, named CustomerDAO and OrderDAO. When we implement the software system in a relational database, these DAO implementation classes are implemented by a group of similar SQL statements such as Select and Insert. When we need to modify these SQL statements, we have to modify all the SQL statements that implement all these DAO implementation classes.

To solve the above problems, we propose a DAO pattern with stronger independency and dynamic extensibility based on the concepts of data binding, meta data and meta model.

2. A New DAO Pattern

The new DAO pattern we propose is shown in Fig. 2.
In Fig. 2, DAOFactory is an abstract class, but not an interface. It implements Singleton pattern[6]. Its method getInstance is shown in List 1. This method helps to load dynamically the subclasses of the class DAOFactory according to the value of the item factoryClass of the system configuration file.

List 1

The method getInstance() of the class DAOFactory

```java
private static DAOFactory m_instance;
public synchronized static final DAOFactory getInstance(){
if(m_instance == null) {
    m_instance = ForClass(System.getProperty("factoryClass")).instance();
}
return m_instance;
}
```

The method createDAO of the class DAOFactory is used to create DAO objects and it is an abstract method.

The DAO interface implements all the access to business objects. It implements CRUD (Create, Retrieve, Update, Delete) pattern[7]. Its methods create(), retrieve(), update(), delete() are used respectively to create, retrieve, update and delete business objects and they are all abstract methods.

The using process of the new DAO pattern[8] is shown in Fig. 3. First, a client calls the method getInstance() of the class DAOFactory to load dynamically an object of the class RdbDAOFactory, which is a subclass of the class DAOFactory. Second, an object of the DAO class, such as the class RdbDAO, is created through the object of the class RdbDAOFactory. Finally, a client can create business objects needed, such as an object of the class Customer, and then the access to the business objects, such as retrieving and deleting, can be implemented through the object of the DAO class. In a word, we can implement all the access to all the business objects through a DAO class.

3. Examples

As Fig. 4 shows, an example is given to illustrate the new DAO pattern.

The class RdbDAOFactory, a subclass of the class DAOFactory, has two private attributes: DataSourceName (standing for the name of the data source) and MapFileName (standing for the name of the mapping file).

The mapping file is shown in List 2. It describes the mapping between business objects and database tables and the mapping between the attributes of business objects and the attributes of database tables. List 2 shows the mapping between the business object Customer and the database table Customer. The business object Customer has three attributes: Name, SSN and Address, which have respectively mapping to the corresponding attributes Name, SSN and Address of the database table Customer. The attribute SSN is the primary key.

According to the mapping file described in List 2, we can create objects of the class ApplicationMap, ObjectMap, PropertyMap through data binding tools such as Castor[9] and Digester[10] when the object of the class RDBDAOFactory is created. The class ApplicationMap encapsulates the mapping information of this application. The class ObjectMap encapsulates the mapping information of business objects. The class PropertyMap encapsulates the mapping information of attributes of business objects. Each object of the class ApplicationMap has some corresponding objects of the class ObjectMap and each object of the class ObjectMap has some corresponding objects of the class PropertyMap.

As Fig. 4 shows, the class RdbDAO is a subclass of the class DAO and it implements methods such as create(), retrieve(), update(), delete(). List 3 describes the implementation codes of the method update().

Finally, we give a part of our application program to illustrate how to use the new DAO pattern, as List 4 shows.
We can give business objects dynamic extensibility with the new DAO pattern. For example, if we want to add the business object Account, we don’t have to modify any codes of the class DAOFactory. All we need to do is to modify the mapping file. Furthermore, because we have only one DAO implementation class to accomplish all the data access to business objects, when we want to modify some SQL statement, all we need to do is to modify the DAO implementation class but not to modify any business objects.

List 3
Codes of the method update()

```java
def update(Object o) throws DAOException {
    //obtaining the object of the class ObjectMap
    ObjectMap objectMap = (ObjectMap)(getDomainMap.get(o.getClass().getName()));
    //creating SQL statements according to meta information of the Mapping Class and the Business Objects
    StringBuffer dmlBuffer = new StringBuffer(“UPDATE”);
    dmlBuffer.append(objectMap.getTableName());
    dmlBuffer.append(“SET”);
    for (Iterator i = objectMap.getPropertyMap(); i.hasNext();)
        PropertyMap propertyMap = (PropertyMap)i.next();
        String value = null;
        try {
            Method method = o.getClass().getMethod(“get” + propertyMap.getAttributeName(), null);
            value = method.invoke(o, null).toString();
        } catch (Exception ex) { throws new DAOException(); }
        dmlBuffer.append(propertyMap.getColumnName()).append(“ = “).append(value);
    //executing the SQL statements that dml specifies
    execute_sql(dmlBuffer);
    return;
}
```

List 4
A part of the application program

```java
DAOFactory daoFactory = DAOFactory.getInstaned();
DAO dao = daoFactory.createDAO();
Customer customer = new CustomerDAO();
CustomerSetSSM ssm = new CustomerSetSSM();
da.create(customer); //creating a DAO object
... customer.setNama(“Peter”);
da.update(customer); //updating a DAO object
... dao.delete(customer); //deleting a DAO object
...
```

4. Conclusions

We analyze the limitations of the generic DAO patterns in the paper, and propose a new DAO pattern with stronger independency and dynamic extensibility based on the concepts of data binding, meta data, and meta model. The structure graph and the sequence graph of the new DAO pattern are given, and then the using process is illustrated through practical examples.

There is still a lot of work to do with the new DAO pattern, such as transaction administration during data access process, complex mapping between business objects and database tables, the performance of the software system, etc.

References


Cheng Fang was born in Hubei Province, China, in 1974. He is a researcher with Department of Computer and Information Engineering, Wuhan Polytechnic University. Now he is pursuing his master degree with Wuhan University. His research interests include software engineering and information engineering.

Ping Zeng was born in Hubei Province, China, in 1966. She is an associate professor with School of Computer Science, Wuhan University. Her research interests include software engineering, information engineering and security of operating system.