USING AN OBJECT-ORIENTED DATABASE TO SUPPORT A WEB APPLICATION BUILT WITH JAVA TECHNOLOGIES

Charles R. Moen, M.S.
University of Houston - Clear Lake
crmoen@juno.com

Morris M. Liaw, Ph.D.
University of Houston - Clear Lake
liaw@cl.uh.edu

ABSTRACT

This paper describes how an object-oriented database can be used to support a web site built with Java technology.

INTRODUCTION

It seems natural to store the data of a web site built with an object-oriented language, like Java, in an object-oriented database (OODB).

This paper describes a web application that uses ObjectStore, a leader in the OODB market [2,4], instead of using a traditional relational database. It discusses the ObjectStore approach to the following technologies: a class file postprocessor, persistence-capable objects, root objects, session and transaction management, and object queries.

A web application called the Online Grade Book was successfully implemented to demonstrate these technologies, and it can be tested at the following link, with the sample Student ID, p001111: http:// oodb-8.cl.uh.edu:8080/OSJIGradeBook/CheckGrades

The following technologies were used to implement the web site: J2EE Servlets and JavaServer Pages (JSP) [3], Model-View-Controller architecture, and J2EE patterns.[1]

THE ONLINE GRADE BOOK

The Online Grade Book is a web application that students can use to check their grades in a particular course. It also has a web-based teacher's interface that is used to enter the student names and grades into an OODB.

The students' interface for the application is shown in Fig. 1. It first prompts the user to enter a Student ID. If that ID matches the password attribute of a Person object stored in the database, the grades for that student will be displayed.

The teacher's interface allows the user to view the grades of all the students stored in the database, edit them, add new students, and change the percentage weights used to calculate the student's overall course score.

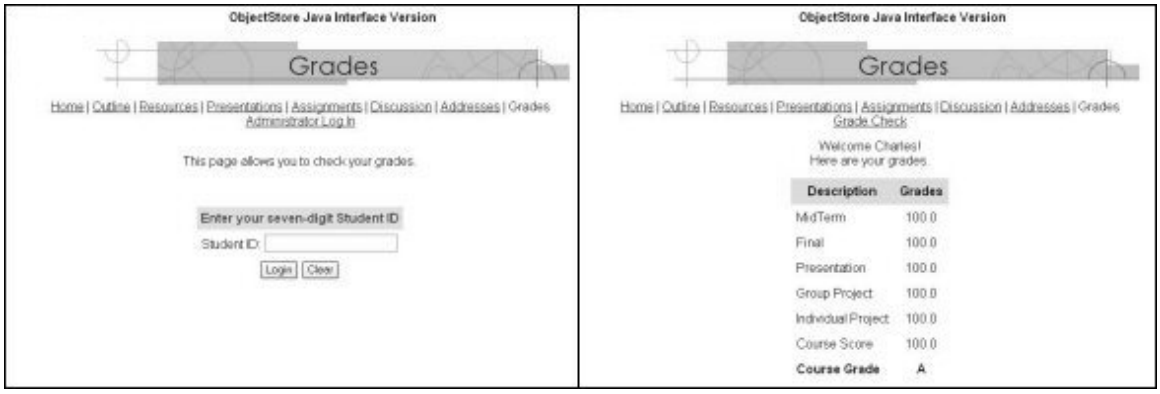

Fig. 1. Left, the Online Grade Book student login. Right, the grade report.

STRUCTURE OF THE DATABASE

The OODB that supports this application stores the student data as objects of a Person class which has attributes for the student's user ID, password, names, and each of the grades earned by that student, as shown in Fig. 2. The methods of the Person object are not shown, and they consist of "get" and "set" methods for each attribute.

Weight objects are stored in the OODB, also. The Weight class is shown in Fig. 2, and its attributes are the grade weights that are used when calculating a student's course score. There is only one object of the Weight class stored in the database for each course.

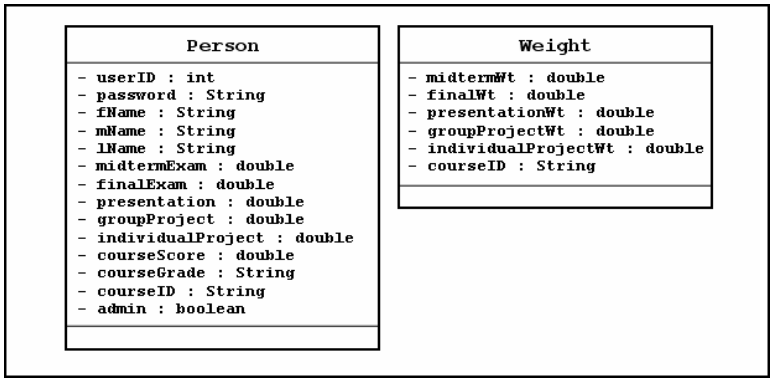

Fig. 2. UML diagram of the Online Grade Book OODB schema

STRUCTURE OF THE APPLICATION

The structure of the Online Grade Book application is based on well-known J2EE patterns [1] such as Business Delegate, Business Service, Data Transfer Object, and Data Access Object. A high-level view of its architecture is shown in Fig. 3.

The application is separated into a web tier and a business tier, so that different hosts can be used for each tier. The OODB resides in the business tier, and Person objects and Weight objects are stored in and retrieved from the OODB by objects of the two data access classes, PersonDAO and WeightDAO.

The data access objects (DAO) are instantiated from the two DAO classes by static methods of the GradeBookDAOFactory class, because the DAO constructors require the URL of an OODB as an argument. The factory class methods read the URL from a properties file before they instantiate a DAO.

The web tier uses the Model-View-Controller architecture [1], where the view is the presentation of data by JSP pages, and a controller servlet manages the logic and responds to the user's actions. The data in the model component is contained in a Person data transfer object, which is retrieved from the OODB by a GradeBookDelegate client when it contacts the GradeBookService using TCP/IP.

The services provided by the business tier are defined in the GradeBook interface. This interface is implemented in the business tier by the GradeBookService class and on the web tier by the GradeBookDelegate class. When a user submits an HTTP request that requires data from the OODB, the GradeBookController calls an appropriate method of the GradeBookDelegate, which then uses TCP/IP to connect with the GradeBookService and invoke its methods to retrieve the data on the business tier.

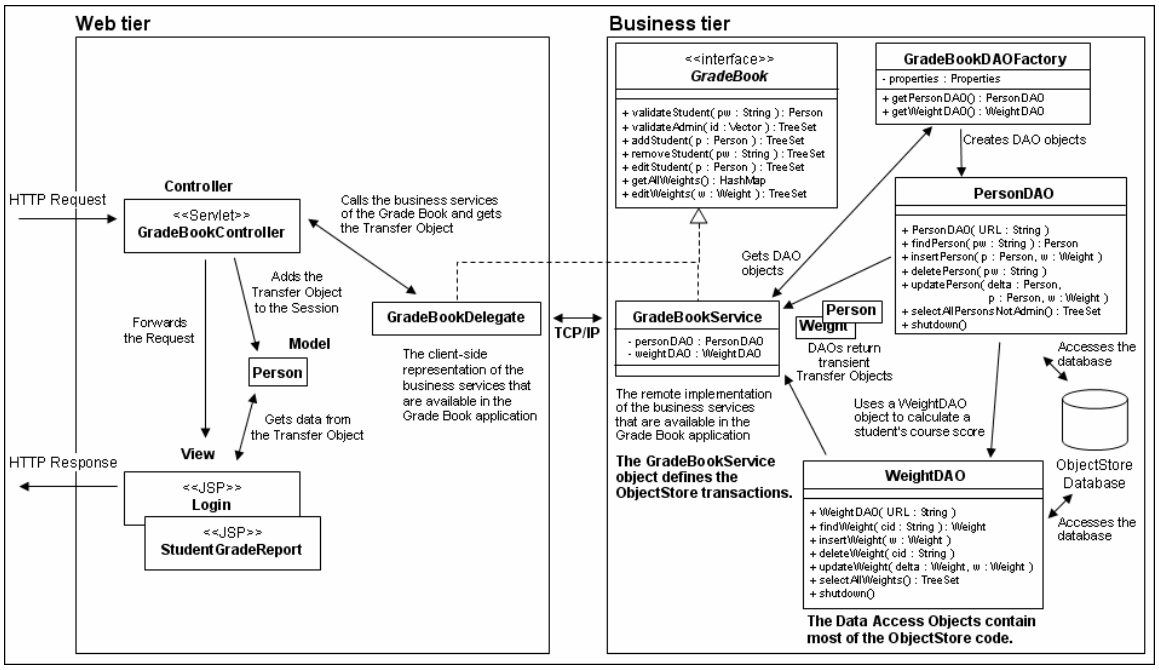

Fig. 3. Structure of the Online Grade Book application

THE CLASS FILE POSTPROCESSOR

Before objects of a particular class can be stored in an ObjectStore OODB, the class must be altered to make it "persistence-capable."[5] ObjectStore provides a class file postprocessor that automates these alterations or "annotations."

To make an ordinary Java class persistence-capable, it must be compiled first and then run through the postprocessor. The postprocessor is executed from the command line, and it can annotate either a single class or a batch of related classes. The annotated class files can then be used to instantiate persistent objects, that is, objects that are capable of being stored in the OODB.

OPENING AND CLOSING THE OBJECTSTORE OODB

All ObjectStore operations, such as opening the OODB, storing an object, and retrieving an object, must take place within an ObjectStore session. So the first step in using an ObjectStore OODB is to open and join a session.

Code that creates a session, joins it, and opens the OODB is shown in the "initialize" method of PersonDAO, Fig. 4. Each of the DAO classes has an initialize method, which is called by the constructor. All the operations for storing and retrieving persistent objects in the Grade Book application are implemented only by methods of DAO classes, so the session has to be opened or joined only when a DAO object is instantiated and used.

In addition, each of the DAO classes has a public "shutdown" method that can be called to close the database and terminate the ObjectStore session.

ROOT OBJECTS

All objects that will be stored in an ObjectStore OODB must have a "root." [5] A root

is a persistent object in the OODB that has a unique name and is created as an "entry point" for persistent objects saved by an application. The root's name becomes the handle that can be used by the application to store or retrieve any particular object.

```
public class PersonDAO{
 private static Session session;
 private static Database db;
 private static OSHashMap allPersons;

 public PersonDAO( String DBURL ){
   initialize(DBURL);
 }

 private void initialize( String dbName ){
   session = Session.getCurrent();
   if( session == null ) session = Session.create(null, null);
   session.join();
   try{
     db = Database.open( dbName, ObjectStore.UPDATE );
   }catch( DatabaseNotFoundException e ){
     db = Database.create(dbName, ObjectStore.ALL_READ | ObjectStore.ALL_WRITE);
   }
   Transaction tr = Transaction.begin( ObjectStore.UPDATE );
   try{
     //Get the root for Person objects stored in the database
     allPersons = (OSHashMap)db.getRoot("allPersons");
   }catch( DatabaseRootNotFoundException e ){
     //The database root was not found, so create a new one
     db.createRoot("allPersons", allPersons = new OSHashMap());
   }
   tr.commit(ObjectStore.RETAIN_READONLY);
 }
 public void shutdown(){
   db.close();
   objectStoreSession.terminate();
 }
 public Person findPerson( String password ) throws DatabaseException {
   Person person = null;
   person = (Person)allPersons.get( password );
   if( person == null ){
      throw new DatabaseException("Could not find person: " + password );
   }
   ObjectStore.fetch(person);
   return person;
 }
 public TreeSet selectAllPersonsNotAdmin() {
    Collection allPersonsValues = allPersons.values();
    Query query = new Query(Person.class, "!isAdmin()");
    Iterator it = query.iterator(allPersonsValues);
    CompareUserIDs comp = new CompareUserIDs();//Sorts Person objects on the ID field
    TreeSet persons = new TreeSet(comp);
    while( it.hasNext() ) {
       persons.add( (Person)it.next() );
    }
    return persons;
 } //...
```

Fig. 4. PersonDAO class

While it is possible to store a single object with a unique root, it is more practical to use a collection object as a root so that many similar persistent objects can be stored in the same root. This approach is analogous to the way a relational database uses a table to store multiple records. ObjectStore provides several persistence-capable collection classes that can be used to create roots, including OSVectorList, OSHashBag, OSHashMap, OSHashSet, OSTreeMap, and OSTreeSet.

The root must be created or retrieved inside a transaction. Once the root has been acquired, the transaction can be committed, as shown in the initialize method in Fig. 4.

In the Online Grade Book application, both the Person objects and the Weight objects are stored in roots created from the OSHashMap class. The OSHashMap object that stores Person objects is named "allPersons," and the password attribute of the Student class is used as the map key. The initialize method in Fig. 4 demonstrates the code to create the "allPersons" root and to retrieve it.

```
public class GradeBookService implements GradeBook{
 private static Session session;
 private PersonDAO personDAO;
 private WeightDAO weightDAO;

 public Person validateStudent( String studentID ){
    Transaction tr = Transaction.begin(ObjectStore.READONLY);
    Person student = null;
    try{
       student = personDAO.findPerson( studentID );
    }catch( DatabaseException dbe ){
       tr.abort(ObjectStore.RETAIN_TRANSIENT);
       System.out.println( dbe.getMessage() );
       return null;
    }
    tr.commit(ObjectStore.RETAIN_TRANSIENT);
    return student;
 } //...
```

Fig. 5. GradeBookService class

RETRIEVING OBJECTS

Persistent objects can be retrieved, inserted, deleted, or updated in an ObjectStore OODB only within a transaction. In the Online Grade Book application, almost all transactions are defined in the GradeBookService class which implements the services provided by the business tier. These services are implemented in the GradeBookService methods by using DAO objects to retrieve data from the OODB, insert it, delete it, or update it.

The code for the "validateStudent" method of GradeBookService, shown in Fig. 5, provides an example of retrieving data from an ObjectStore OODB within a transaction. This method validates a student who entered a Student ID in the text field of the entry page of the Online Grade Book application. The transaction is opened at the beginning of the method body in "READONLY" mode, and then the ID is passed to the "findPerson" method of the personDAO. The findPerson method, see Fig. 4, looks for a Person object stored in the "allPersons" root that has a key value that matches the ID. If the key exists in the database, then the Person object stored with that key will be returned; else an exception is thrown by findPerson. If the validateStudent method catches this exception, it aborts the transaction and returns null; otherwise it commits the transaction and returns the Person object that was retrieved by findPerson.

When using an ObjectStore OODB, it is important to realize that when an object is retrieved from the OODB, there are two copies of it, one in memory, which can be used by the program, and the other, which is still on the disk. ObjectStore connects these two copies so the copy in memory will always have access to the stored copy's current state.

When an object is first retrieved from the OODB, the in-memory copy is in a state

called “hollow.” A hollow object contains no data, and its attributes contain only “null” values. As long as this hollow object is used within transaction boundaries, then the data will be fetched from the matching database object automatically whenever it is needed. A hollow object’s data can also be explicitly retrieved by calling the static ObjectStore “fetch” method, thereby making the object’s state “active.” The findPerson method explicitly calls “fetch” to fill the data fields of the Person object before it is returned, as shown in Fig. 4.

In addition, before an active persistent object can be returned to the web tier where it will be used outside the boundaries of a transaction, it must be made “transient.” Transient objects have no connection to the ObjectStore OODB and can be used like ordinary objects. This conversion from an active state to a transient state is made by committing the transaction with the RETAIN_TRANSIENT mode.

OBJECTSTORE QUERIES

Queries can be used to retrieve data from an ObjectStore OODB, and the PersonDAO method named “selectAllPersonsNotAdmin” illustrates one use of the query syntax, see Fig. 4. This method uses a query to retrieve all the students in the OODB by selecting all Person objects that have “false” as the value of their admin attribute.

While a query cannot be run on an OSHashMap object, it can be run on a Collection. So this method begins by creating a Collection containing all of the objects stored in the allPersons root. It then instantiates a query object by specifying two arguments in its constructor; the first argument is the class of objects that the query will work with and the second argument is an expression that returns a Boolean value. After the query object is instantiated, the query is run by calling its “iterator” method and passing it the Collection as an argument. The iterator method of the query object returns an Iterator object containing the objects that are the result of the query.

CONCLUSION

The Online Grade Book successfully demonstrates how an object-oriented database can be used to support a web application built with Java technologies.

REFERENCES

1. Alur, D., Crupi, J., Malks, D., *Core J2EE Patterns: Best Practices and Design Strategies, 2nd edition*. Upper Saddle River, NJ: Prentice Hall PTR, 2003.
2. Bloor Research, ObjectStore from Progress Software, http://www.objectstore.com/docs/analyst/bloor_0105_objsov.pdf, 2005.
3. Hall, M., *Core Servlets and JavaServer Pages*. Upper Saddle River, NJ: Prentice Hall PTR, 2000.
4. Lamb, C., Landis, G., Orenstein, J., Weinreb, D., The ObjectStore Database System, *Communications of the ACM*, 34(10), 50–63, 1991.
5. Progress Software Corporation, ObjectStore Java Interface Release 6.1 Bookshelf, http://www.objectstore.net/documentation, 2003.