EOF-Overview

WebObjects has two facets. One facet takes a relational database, or some other persistent data source, and turns it into an object graph, a group of objects that connect to each other. Another facet takes an object graph and uses templates to produce HTML. The first facet is EOF. EOF bridges the world of relational databases to the world of objects. To put it in other terms, EOF allows you to start with a database and build POJOs (POJO = "Plain Old Java Object") that represent the things you want to do on top of your database, to make it smarter and more responsive to your desired audience.

If you have come from a relational database environment, you may have been trained to think in terms of tables, rows, columns, joins, Cartesian cross products, result sets and the like. So long as you continue to view your applications in those terms, WebObjects will stubbornly stand between you and success by imposing an extraordinary number of "conveniences" in your way to a quick SQL solution to your immediate problem. The schema design under your application is important, but the schema design is not the same as the database design. EOF gives you a layer of abstraction over a simple relational database.

If you can wrap your mind around the object-oriented concepts, using the object-oriented collection classes, such as arrays (or vectors), sets, dictionaries (or hash tables) and think in terms of the object graph with which you're working, you will find WebObjects to be a very supportive and helpful technology. Like all complex technologies, it has its glitches and bugs, but they are surprisingly few for the incredible complexity underlying the surface.

There are two ways to approach using EOF. It is used both as a way to make existing databases more flexible or as a way to perist models in applications being developed. It is used both by entrepreneurs who model on a whiteboard first and write code last, and by companies with 15 year old databases and legacy code and systems that have to be made more flexible.

If you already have a database, the best way to get started is to reverse engineer a model. If you have WebObjects on a Mac OS X system, you have an ex ample database and a JDBC driver for it.

The best way to get a handle on what WebObjects may be for you is to take a long hard look at your EOModel as an ER diagram. Open your EOModel in EOModeler and switch to the graphic Entity-Relationship view (the little pop-up in the upper left corner of the EOModeler window). That view represents the object graph of your model objects. Rearrange the entities by dragging to get as little line crossing as possible so you can see the diagram clearly. Once you've done that, you might want to print it out to make access more convenient. An easy way to do this, if you're on a Mac, is to take a screen shot of it and then you can print out the resulting .pdf file and have it scaled to a single sheet of paper.

Note that the names of the attributes in each of the entities on that diagram are the official EOModel "attribute" names rather than the database column names. EOModeler will create a java class file for each of those entities, if you ask it to, and the java member names (ivars and method names) will correspond to those names as well. This is the first step to thinking in the WebObjects way. Think in terms of those official "attribute" names for any operation rather than the database column names.

Once you have wrapped your mind around the model, the next step is getting some data. Pick an entity in which you're interested. To instantiate a set of objects in your application for that entity, you've typically got to create an EOQualifier to select the appropriate instances from all the instances that are out there in your object store (notice I said object instances, not table rows, and object store, not relational database; gotta get used to thinking objects).

Say you have a set of Employee and Department records out there and you want to get all the Employees who work for your department to send them invitations to the division end-of-year-blow-out (EOYBO). Further, assume that your Employee and Department entities have the following attributes:

```
Employee
  lastName
  firstName
  lengthOfService
  ...

Department
  name
  location
  division
```

If you just wanted employees of a given gender, say, "female", the easiest way to get them would be to use EOUtilities as a handy short cut for easy queries.

```
myEditingContext  = session().defaultEditingContext();
NSArray femaleEmployees = EOUtilities.objectsMatchingKeyAndValue(myEditingContext,
                            "Employee", "gender ", "female");
```

There's a lot there, but you can look up the method in the documentation for EOUtilities to see what the arguments mean. In any case, once this statement is executed, you have an NSArray of the employees you're looking for. (look NSArray up too and compare it to Java's Array or Vector which are similar). Unfortunately, just comparing to a given attribute by a specific value is a bit confining, so the longer, but more flexible, way is to create an EOQualifier, then use it in a fetch specification to get your initial objects. The easiest way to create your qualifier is with the magic method, "qualifierWithQualifierFormat()". With these tools, we can get the employees we were first looking for, that is, those in our department. Let's do that like so:

```
myDepartmentName = "Information Technology";
EOQualifier myQual = EOQualifier.qualifierWithQualifierFormat("department.name = %@", myDepartmentName);
EOFetchSpecification myFetcher = new EOFetchSpecification("Employee", myQual, null);
EOEditingContext myEC = session().defaultEditingContext();
NSArray employeesToInvite = myEC.objectsWithFetchSpecification(myFetcher);
```

Once again, we end up with an array of Employees but using a slightly harder approach. Notice, however, that we've asked the database to do a join for us by using that little "department.name" construct in the qualifier format. That says, get the employee connected to a department object whose name is... The advantage, of course, is that the magic method, qualifierWithQualifierFormat() allows us to build some pretty complex qualifiers, and allows us to do so easily, such as:

```
EOQualifier myQual = EOQualifier.qualifierWithQualifierFormat(
   "(department.name = %@ and age > %@ and boss.gender = %@) or gender = 'female'",
   myDepartmentName, new Integer(40), "male");
```

The hard part is done. Now we only have left the stuff that WebObjects makes so easy for us as an object-oriented programmer. Suppose that for each of these employees, you wanted to schedule them for the EOYBO by sending messages to their bosses. Presume also, that bosses, like employees, are Employees, so have the same attributes shown above. Simply iterate through your array of Employees and ask each object for its boss's email address and send off the invitation, like so:

```
Enumeration enum = likelyEmployees.objectEnumerator;
while (enum.hasMoreElements()) {
    Employee anEmployee = enum.nextElement();
    sendScheduleRequestTo(anEmployee.boss().emailAddress());
}
```

Notice that we didn't need any more fetch specifications or qualifiers! WebObjects, through its Enterprise Objects Framework, automatically recognizes the need for more records from the database (the bosses) and just goes and gets them for us without the need for more explicit fetching.

Granted, someone would have to write the method, "sendScheduleRequestTo(EMailAddress addr)", but there are lots of libraries available, and you probably know Java well enough to do that yourself.

The point is, we don't have to do an extra explicit fetch for the bosses because WebObjects does it for us. It basically does the join operation and provides us the result invisibly. Just get your first set of objects with a fetch, then follow the object graph.

Forget result sets, joins, selects and all the rest of those relational database artifacts. Or, at least, forget them till you get in trouble, somehow, obtaining the results you're looking for. If the trouble is deep enough, set the WebObjects runtime parameter, -EOAdaptorDebugEnabled YES, and the EOAdaptor will dump all the SQL it generates out onto your run log, so you can see exactly what it's doing for (or to) you. 🙂

Before you go off and build your first critical, human-life-hangs-on-no-bugs WebObjects program, however, you should have a look at "General Guidelines for Managing the Object Graph". This goes a little deeper into some of the concepts I've referenced here and is written very well.

You can find it in the Apple documentation at: `/Developer/Documentation/WebObjects/Reference/API/com/webobjects/eocontrol/concepts/EOEditingContextConcepts.html#GENERALGUIDELINES`

or

`/Developer/ADC Reference Library/WebObjects/Reference/API/com/webobjects/eocontrol/concepts/EOEditingContextConcepts.html#GENERALGUIDELINES`

That, of course, is an HTML fragment of a much larger page, all of which is worth reading.

In any case, good luck with your WebObjects future.

Class References:

- http://wocommunity.org/documents/javadoc/WebObjects/5.4.2/com/webobjects/eoaccess/EOUtilities.html