Author: Bill Ennis

TOPLink provides container-managed persistence for BEA Weblogic. It has been available for Weblogic's application server since Weblogic version 4.5.1 released in December, 1999. TOPLink can also be used in a pure java environment to access most JDBC enabled relational databases; i.e. you can use it to communicate between java classes not deployed to an application server and a relational database. TOPLink includes the following features:

Feature ***Description*** **Object Oriented API for object storage** Programmers use methods readObject() and writeObject() to interact with the database. TOPLink translates into the necessary SQL to carry out the operations that may require to several underlying database tables.

Manage complex relationships between objects

Supports 1-1, 1-many, and many to many relationships between objects

Indirection

Indirection defers database access required by dependent objects. For example, the phone information for an Employee object.

Optimistic Locking

Allows for greater system concurrency since database locks are held only as needed.

Object Query API

Programmers construct Expression objects. TOPLink translates these into the necessary SQL.

Client/middle tier object cache

Objects are cached to reduce database access

Support for inheritance

Classes using inheritance may be “mapped”

Performance features

Includes batched writing, table-joined reading, advanced cursor structures

Type conversion

Automatic conversion of java to SQL data types

Multiple database support

If JDBC driver is supplied TOPLink can support the underlying database

Container managed persistence for WebLogic and WebSphere Application Servers

Works as a container in an EJB environment as a substitute for the vendors own Container for persistence.

Table 1: TOPLink features

A brief Definition of Object-Relational Mapping

The basic problem that any O/R mapping tool attempts to solve what is known as the impedance mismatch between an object model and a database schema. Object models often use relationships between objects that introduce a hierarchy. These relationships introduce complexity when interacting with a relational database since the construction and storage of an object will then be accomplished by interacting with several database tables, rows, and columns. A simple example would be an object that represents a person who in turn can have multiple phone numbers. This would typically be represented in an object model as a person object containing a collection of phone objects (see Figure 1). In a data model would be a person table, a phone table, and a foreign key relationship between those tables. Whenever you construct a person you would need to gather the information from the tables and set the appropriate attributes in the person class. This would be accomplished via either a database join or a series of database accesses with accompanying code to read the data into program variables and move to the appropriate object attributes. Object Relational Mapping Tools/libraries encapsulate the details of how a database schema is translated into an object model and also provide the data access layer. Because of this you are dealing with normal java objects within your code with object oriented method calls such as readObject() and writeObject() when you wish to persist or instantiate an object graph. In addition each time you create a new query (i.e. a unique where clause) you usually need to add some code to perform this new operation. With an O/R mapping tool there is still some work to add new queries, but this work is isolated to the definition of the new query and does not include the cursor

manipulation processing to gather the information (which is sometime a substantial amount of code). See figure 2 for come some source code related to reading and writing objects with TOPLink. If you use TOPLink for container-managed persistence within an application server you can avoid having to even write access logic. TOPLink implements the ejbLoad and ejbStore methods that correspond to reading and writing objects.

Figure 1: Person Object representation vs. Relational Database representation

```
package TOPLink.Tutorials.Benchmark;

import TOPLink.Public.*;
import TOPLink.Tools.SchemaFramework.*;

  protected Session session;
public Project project;

  TOPLink.Public.PublicInterface.Project builderProject;
// read the TOPLink project file. This is a Table of Contents for all TOPLink metaData
builderProject = TOPLink.Public.PublicInterface.Project.read(
&quot;C:\TOPLink\Benchmark\Benchmark.project&quot;);
// Databases Session is a TOPLink class
session = new DatabaseSession(builderProject);
session.login(&quot;bille&quot;, &quot;bille&quot;); // database user and password

  // create and Person object
Person person = new Person();
person.setFirstName(fname);
person.setLastName(lname + &quot;-&quot; + iter);

  Vector phoneVec = new Vector();
for (int i=1; i<=5; i++) {
Phone phoneNum = new Phone();
phoneNum.setAreaCode(areacode);
phoneNum.setPhoneNumber(prefix + &quot;-&quot; + iter + &quot;-&quot; + i);
switch (i) {
case 1:
phoneNum.setType(&quot;Desk&quot;);
break;
case 2:
phoneNum.setType(&quot;Cell&quot;);
break;
}
```

```
phoneNum.setOwner(emp);
phoneVec.addElement(phoneNum);
}
person.setPhoneNumbers(phoneVec);
dbSession.writeObject(emp);
Expression persExp = new ExpressionBuilder().get(&amp;quot;empId&amp;quot;).equal(1);
Person person = (Person) benchMark.session.readObject(Person.class, persExp);
session.logout();
```

Figure 2: Sample code using TOPLink to write and read objects

Some other advantages of defining a persistence layer is that it allows a uniform representation of the data model to the application, and a client side object cache. When an application creates multiple access points to the same data element it becomes difficult to maintain when database schema changes are introduced. This is because the change must be propagated across all access points that are affected by the change. Client side object caches improve performance since they can reduce database accesses. One must keep in mind that this layer exists however whenever dealing with the database directly. For example, you may perform a direct SQL update on a table without the persistence layer. Since the data access layer does not know about this update it may not reflect the update in its cache (unless refreshed).

Personal history with O/R Mapping Tools

In June of 1998 I was asked to evaluate Object Oriented Databases. My early findings were encouraging. The OO database is was experimenting with seemed to have most of the operational functionality of many of the mature relational databases being used. It was at this point I was handed the task to kick the tires. The first thing I did was write a simple java application to load data into my database. I was impressed with the java database interface in it simplicity to use and was able to quickly load 1GB of data. Then it was time for some operational tests involving backup/recovery and replication. This is where the product came up short. The database core dumped during backups – maybe they never tested on a 1GB database? Creating a replica of my database never seemed to end. After stripping the database down to 50 MB I was able to create a replica in 27 hours! I didn't need to look any further to see that Object Oriented databases were not ready for prime time. Therefore the focus was put on Object Relational Mapping technologies, since we would be using java and a relational backend (Oracle and Sybase were the preferred vendors at the site I was working at).

We evaluated several including several application servers that included O/R mapping technology. We found that those products that focused solely on object-relational mapping

always had the best feature set. TOPLink was selected as the technology based on the examination of its feature set and ease of use/speed in getting something up and running.

Since O/R mapping is a desired feature in a java application server ObjectPeople (the makers of TOPLink) have started to provide container-managed persistence for some of the major application server vendors (the first being BEA Weblogic). Weblogic includes basic O/R mapping functionality, but it is limited with regards to relationships between objects, cache configurability, defining queries, and tools to support modeling of the mapping.

Case Study – workzone360.com

Workzone360.com is using an EJB (Enterprise Java Bean version 1.1) based architecture. The application server used is Weblogic version 5.1. The architecture allowed us to take advantage of container-managed persistence. Therefore TOPLink was an immediate candidate. Since we had some experience on the team with it and it could potentially save significant time we decided on TOPLink.

Installation

The installation of TOPLink is not automatic. There are some subtle things that you need to be aware of. There is a jar file that comes with the software that must be copied to the weblogic directory hierarchy. This jar file (TOPLink_CMP.jar) needs to be in your $WEBLOGIC_DIR/lib/persistence directory. The install attempts to put this in there for you but isn't always successful do to several possible scenarios. In addition, you must modify CLASSPATH's to include the other TOPLink jar files. The TOPLink manuals actually cover the classpath information is adequate detail. However, those accustomed to NT based installs are not used to having to make these types of changes after an install. TOPLink licensing is developer based (no runtime fees apply). License keys need to be entered into your weblogic.properties file with the property

Java.system.property.toplink.license='<YOUR_TOPLINK_KEY_HERE>'

Figure 3: weblogic.properties file addition for TOPLink license key

You also need to enable access for java reflection on our classes from within the weblogic

policy file.

```
Grant {
Permission java.lang.reflect.ReflectPermission &quot;suppressAccessChecks&quot;;
Permission java.io.SerializablePermission &quot;enableSubstitution&quot;;
Permission java.util.PropertyPermission &quot;*&quot; &quot;read,write&quot;;
}
```

Figure 4: weblogic.policy file entries added

The TOPLink Project file and related files

The TOPLink Builder organizes the files it creates/edits with the use of a project file. The project file is the table of contents for a series of files that are used by the TOPLink Builder and eventually by the TOPLink container when your Entity Beans get deployed. Each Entity Bean class will have a file that describes the class (ending in .topclass). Each database table involved in a mapping will also have a designated file (ending in .table). And the mappings between classes and tables are stores in another file (ending in .descriptor). TOPLink often refers to a mapping as a descriptor. A descriptor contains information about a class attribute and what database table column that it maps to. TOPLink also supports generation of the mappings into java class files for production deployments.

It is also possible to use TOPLink to map regular java objects that are not Entity Beans. These are usually dependent objects that you do not believe need the functionality (and excess baggage) of an Entity bean.

EJB XML Deployment files

In workzone360 we decided to deploy all beans in a common jar file. Each EJB deployment requires a set of XML files that define container dependent behavior for those beans. EJB 1.1 requires XML deployment descriptor files. Both Weblogic and TOPLink supply dtd's for the respective xml deployment files. These files include:

- Ejb-jar.xml – This file identifies the beans to be deployed, their persistence type (TOPLink in our case), along with any transactional and security characteristics that will be attached to the bean methods. The file is separated into two major areas of enterprise beans and their

associated assembly descriptors. The enterprise bean section is basically a listing of the beans to be deployed. The assembly area allows the bean deployer to assign transactional and security characteristics at the bean method level.

- Weblogic-ejb.xml – This file again names each bean and defines associated caching behavior, and persistence definition. Each entity bean using TOPLink will declare so within the persistence-descriptor and will point to an associated TOPLink cmp file.
- Toplink-cmp-<bean>.xml – You will wind up with one of these files for every bean that you have mapped within TOPLink. Within the file you will point to a TOPLink project file (discussed previously in this document), assigned a weblogic jdbc connection pool, and define various finder methods (i.e. queries) for the bean.

Project Outcome

We were able to achieve significant success with TOPLink on the workzone360 project. We did not have a need to create any SQL due to limitations in TOPLink. The realization of the power of the tool came when we needed to define a new finder for TOPLink mapped beans. Finders are EJB lookup methods, they allow an EJB client to pass in search parameters. Normally, we would have had to create a new interface for the finder and also the underlying data access code to create the query, fetch the data, and construct a data structure to send back to the client. All that was required with our application was to define a new finder (in the TOPLink deployment descriptor file); we did not even have to write code. Below is a cutout of the finder declaration that was required to retrieve all business objects that match a partial business name:

```
<finder>
<method-name>findByPartialBusinessName</method-name>
<method-params>
<method-param>
<param-type>java.lang.String</param-type>
<param-name>businessname</param-name>
</method-param>
</method-params>
<finder-type>EXPRESSION</finder-type>
<finder-query>

<![CDATA[builder.get(&quot;businessname&quot;).likeIgnoreCase(businessname+&quot;%&quot;);]]>
</finder-query>
</finder>
```

We added many finders after mapping our objects and were able to implement data access in a

very productive manner.

Some TIPS worth mentioning

- You do NOT need to reconstruct the deployment jar file in the event that you have changes to previously existing TOPLink project file, descriptor files, etc. The toplink-cmp-<bean>.xml file points out to the project file and the project file is not included in the jar.
- The primary key class of your bean must be public. If this is not the case you will get strange messages that you do not have a primary key named “id”. This is because if the attribute is declared private it will be unavailable to TOPLink and Weblogic.
- Use the jikes compiler. It will speed up compilation and ejbc utility.
- Make sure you configure adequate numbers of connection to your jdbc connection pool. You can run into problems if you only have one or two and get into a situation where you need parallel transactions running.
- Carefully think about the caching strategy that would like to use for each EJB. TOPLink supports several caching strategies. The default is to grow the cache without limit. This is not optimal in most scenarios. You can set caching strategy at a global/project level and override cache settings at the bean level where you wish to deviate from your global setting.
- Consider Indirection – Indirection allows you to employ lay instantiation of objects in an object graph. A great example would be a bill of materials object graph. These can get quite large and constructing the entire graph up front may not be desired. With indirection the database access is delayed until the point that the object is traversed through an accessor method.
- You do NOT have to make entity beans out of every class that you would like to persist to the database. You can map normal java objects as well as entity beans in the same application jar to be deployed in weblogic. Entity Beans should be business entities.

Some Quick Notes on Entity Bean Development with Weblogic & TOPLink

1. Create the Remote Interface and Home Interface of your Bean, and then implement the logic for them in the bean class itself. You should also create public accessor methods for any attributes that you need to persist in the database. Make sure that all Entity Beans contain a zero argument constructor.
2. Create database table that will contain the bean information.
3. Compile the Bean class.
4. Bring up the TOPLink Builder and make sure the new class is in your classpath as well as any classes that they reference. The TOPLink builder is a GUI that allows you to define mappings. It can reverse engineer compiled java .class as well as database tables. After classes and tables are imported into the builder mapping can be performed. When you save the TOPLink project the project file is updated along with definitions for all classes, tables, and mappings between them get saved to files on the file system. The builder associates database connection information with a project along with the JDBC driver to be used.

note: If you ever modify the class you need to consider reverse engineering it into TOPLink.

This comes into play if a getter or setter ever goes to call an internal method of your code.

What does it cost?

TOPLink only sells development licenses (no runtimes apply). You pay by the number of seats. Whoever is developing EJB's on your project should be licensed. TOPLink also has a java only project that is licensed the same way. The cost is in the area of $5000 and discounts start applying at 10 seats on up.

Summary and conclusion

The goal of this paper was to introduce TOPLink as a possible alternative as a persistence layer for java and for BEA Weblogic Enterprise Java Beans. It has also defined the Object Relational impedance problem and discussed some of the work that has been done in this area. More information on TOPLink can be found at http//:www.objectpeople.com. More Information on Weblogic can be found at http://www.beasys.com. Both site maintain a variety of newsgroups with many competent developers offering good advice. TOPLink also can be used in a non-EJB java environment for persisting objects to a relational database.