THE OBJECT PEOPLE
Your Source for e-Solutions

Whitepaper

TOPLink for WebLogic®

The Challenge:

Enterprise JavaBeans (EJB) represents a new standard in enterprise computing: a component-based architecture for developing and deploying distributed object-oriented applications in Java.

Object technology has become the solution of choice for building enterprise applications. However, many organizations have a great deal invested in relational databases.

Relational databases are mature and their capacity and performance are predictable and reliable. Consequently, Java developers often need to access data stored in relational databases. However, the object world and the relational world do not completely match up; one world consists of tables, rows, columns, and foreign keys, while the other contains object references, business rules, complex relationships, and inheritance. This is often referred to as the object/relational "impedance mismatch" and is a major challenge for organizations adopting object technology.

The Solution:

TOPLink for Java answers the challenge by providing a bridge between the Java object and relational database worlds, and by providing comprehensive support for Enterprise JavaBeans (EJBs). TOPLink for WebLogic extends this support to provide container-managed persistence for Entity beans deployed within BEA's WebLogic® Server.

TOPLink for WebLogic handles:

- ✓ Enterprise JavaBeans
  - ✓ session beans
  - ✓ entity beans
- ✓ Persistence
  - ✓ bean-managed
  - ✓ container-managed
- ✓ Java Transaction Service (JTS) integration

Overview

TOPLink for WebLogic is seamlessly integrated with the WebLogic application server from BEA WebXpress, one of the leading Java application servers.

BEA WebLogic provides a comprehensive implementation of the Java Enterprise Standards, including a full Enterprise JavaBeans™ implementation. With TOPLink for WebLogic, you can build effective, efficient components for server-side Java applications, while significantly cutting development time and expense.

TOPLink offers a comprehensive feature set that has evolved based on feedback from thousands of developers in more than thirty different countries. The TOPLink client base reflects a wide cross-section of industries and diverse requirements. As a result, TOPLink has a solid design that combines flexibility and performance. We bring this knowledge and experience to TOPLink for WebLogic.

TOPLink for WebLogic's features include:

- ✓ Enterprise JavaBeans support -- session beans; entity beans using bean-managed and container-managed persistence
- ✓ Java Transaction Service (JTS) integration
- ✓ mapping flexibility with 15 different types of mappings
- ✓ TOPLink Builder, to map objects visually
- ✓ proxies for "just in time" reading
- ✓ object caching and identity
- ✓ object level transactions with units of work
- ✓ batched reading / writing
- ✓ object level queries
- ✓ stored procedures and custom SQL
- ✓ object locking -- optimistic and pessimistic

The TOPLink technology has been in use since 1992 and was first released as a product in 1994. Based on this technology, TOPLink for Java was first made available in 1997 and has quickly become an industry-leading persistence solution. It provides a robust, feature-rich framework that can be reused on multiple projects. TOPLink is used worldwide in industries such as insurance, banking, transportation logistics, manufacturing, aerospace, automotive, system integration, pharmaceutical and health care.

Enterprise JavaBeansÔ (EJB)

Enterprise JavaBeans (EJB) represents a new standard in enterprise computing. Developed by Sun Microsystems and its partners, EJB provides a component-based architecture for developing and deploying distributed object-oriented applications in Java.

It is important to note that EJB is, itself, not a product. Enterprise JavaBeans is a specification (currently at version 1.1) that describes a framework for developers to use to create distributed business applications and for vendors to use to design application servers. Sun Microsystems has partnered with industry leaders such as BEA WebXpress to specify the EJB framework.

What is an Enterprise JavaBean?

To quote the EJB specification, “an enterprise Bean implements a business task, or a business entity”. EJBs are server-side domain objects that fit into a standard component-based architecture for building enterprise applications using the Java language.

The fundamental goal of EJB technology is gains in productivity. Application developers using EJB components benefit because they now have a standard, well-specified way of taking advantage of robust pre-built server components. Developers of server components also gain, because distribution, security, threading and process control, state and resource management, and transactional control are all taken care of by the EJB server.

EJB architecture

An enterprise bean resides on an EJB server within an EJB container. The server provides services such as naming, security, thread management, and transaction control that the container in turn provides to the bean. The container is not an actual component, but represents a logical separation of responsibility between the container and the server. The container wraps the bean such that clients cannot manipulate the bean directly. This allows the container to implement transaction, security, and persistence logic that are distinct from the domain logic of the bean itself.

EJBs and relational databases

EJB, like Java itself, is based on object technology, but EJB applications often need to access relational databases. The difficulties in mapping objects to a relational database are well known and often referred to as the "impedance mismatch". This can be a major issue when adopting Java and EJB technology.

The EJB specification provides an outline of how to handle persistence, but it only covers mapping to very simple data. More complex applications will need to manage bean relationships, beans which map to multiple tables, inheritance, object faulting (just-in-time reading), and advanced querying.

These issues are addressed by using a sophisticated persistence product such as TOPLink.

(For more information on impedance mismatch and mapping, see the “TOPLink for Java” white paper.)

But I’m using JDBC already . . .

JDBC defines a low-level API for accessing databases. It does not work at the level of Java objects. Developers using JDBC must write methods that contain SQL statements and convert between database rows and objects. Application developers work with objects using TOPLink, rather than with rows and SQL using JDBC calls. TOPLink does not replace JDBC, but provides a layer on top of it.

Persistence framework: buy or build?

Building a custom persistence framework can easily consume 30-40% of a project’s resources. This problem is much more challenging than it first appears and often requires the efforts of the most experienced members of a project team. The resulting home-grown framework requires support and maintenance and may not be reusable on other projects. Persistence is often on the critical path of a project, so that a mature, commercial product such as TOPLink provides substantial benefits. With TOPLink, a project’s resources can be focussed on building the application, not on infrastructure.

TOPLink: Advanced object-to-relational technology

TOPLink for Java is an advanced object-to-relational persistence framework. It allows application developers to access data stored in relational databases as Java business objects or Enterprise JavaBeans. With TOPLink, developers focus on the application and object model rather than on the database. TOPLink can map Java objects to an existing legacy database or can generate a new database schema from an object model. It provides a rich set of features to read, write, delete, and manage objects in an efficient manner.

TOPLink uses meta-data "descriptors" to define the correspondence between Java objects and the relational tables used to store them. TOPLink uses these descriptors to dynamically generate the required SQL statements at run-time. The mappings can be changed without having to re-compile the classes they represent. No SQL programming is required.

Scalability and maintainability

It is important for both scalability and maintainability that the persistence layer isolate business programmers from the database. Application developers should not write and maintain SQL statements, but instead focus on the business rules and application logic. TOPLink provides such encapsulation, allowing an architecture where the user interface and business objects no longer depend on where and how the data is stored.

TOPLink supports arbitrarily complex models and automatically maintains references between objects. Changing the database schema does not normally require changes to the business objects, only to the TOPLink mappings. Java business classes can be reused with completely different database schemas. This fully realizes the promise of container-managed persistence by supporting rich, complex models with fully automatic persistence.

Object-level interface

TOPLink provides a level of abstraction that allows the application developer to work with objects. No SQL or JDBC programming is required.

Non-intrusive

TOPLink uses a meta-data model and attempts to leave the Java business classes as “pure” as possible. TOPLink does not force the application developer to pollute their classes with special persistence methods, nor does TOPLink force a designer to extend a `PersistentObject` class. No database-specific methods are generated or hard-coded on the business classes.

A descriptor is defined for each Java business class and TOPLink uses these descriptors at run-time to read, write, and manage the business objects.

The de-coupled mechanism allows a single business class to have multiple descriptors defined for it. The same object can be stored on different databases by changing the descriptor used, without requiring changes to application-level code. With EJBs, this is essential to support third-party beans for which no source code is available.

Iterative development process

Object-oriented development is very iterative in nature. TOPLink’s architecture allows an object and data model to change and evolve without having to regenerate the object/relational mapping information every time an unrelated change occurs.

An architecture that forces the developer to implement methods on the domain classes or that generates a static set of mapping code is more restrictive. The continuing evolution of a complex business object model can neither accommodate the inflexibility of forcing the use of a persistent super-class, nor the burden of maintaining additional special persistence methods.

Performance

TOPLink is designed to minimize the most expensive operations in a multi-tier enterprise application, namely database calls over the network. Several features such as caching, batch reading, cursored streams, batch writing, and pre-allocated sequence numbers are available to optimize performance.

TOPLink’s “unit of work” feature ensures that minimal updates are done. The unit of work can determine what attributes of an object have changed, and write out only those fields.

Platform and database independence

Wherever Java runs, so does TOPLink – its class library is certified 100% Pure Java™. No platform-specific libraries or DLLs are required. TOPLink is supported for both JDK 1.1.x and Java 2. All relational databases accessible with a JDBC driver can be used with TOPLink, including Oracle, Sybase, DB2, SQL Server, Informix, and Access.

Flexibility

TOPLink allows the application developer to control almost every aspect of the system. For example, TOPLink normally generates dynamic SQL, but it may be desirable to use database stored procedures. TOPLink allows the developer to override any generated SQL call with a stored procedure or custom SQL.

TOPLink expands on the EJB-supported callbacks, adding a wide variety of persistence-related events for which an application can be notified. These include pre- and post-read, write, transaction commit/rollback, lookup and delete events.

TOPLink provides many additional features than can be tuned to a variety of architectures. Other features that have variable settings include cache types and size, connection pooling, sequence number allocation, page sizes, database transactions, batch reading, object faulting, and writing depth.

TOPLink for Java, version 2.5

TOPLink for Java, version 2.5, was released in December 1999. For more details, please refer to the TOPLink for Java white paper.

TOPLink for WebLogic

TOPLink for WebLogic provides developers with the advanced object/relational mapping features of TOPLink for Java, integrated with the powerful application server environment of BEA WebLogic.

TOPLink for WebLogic goes further than what is outlined by the EJB specification, providing additional value. Developers of sophisticated enterprise applications can leverage the benefits of both EJB and relational database technology. Using TOPLink for

WebLogic, developers can take advantage of the features available in TOPLink for Java, version 2.5, as well as the advantages of EJB.

TOPLink for WebLogic provides a seamless transparent persistence layer. Features such as WebLogic's distributed Java Transaction Service (JTS) and connection pooling are integrated with TOPLink. The two products are complementary and together offer developers a powerful environment to build and deploy EJB applications.

The flexibility provided by TOPLink carries over to the EJB environment. A bean can be mapped to different database schemas without changing any bean-level methods simply by changing TOPLink descriptors.

There are several ways in which a developer can use TOPLink for WebLogic in an EJB environment. A number of these are described in the following sections.

Session beans

Session beans represent a business operation, task, or process. Although the use of a session bean may involve database access, session beans are not in themselves persistent; they do not directly represent a database entry. Examples of session beans include a "shopping cart" bean or a "bank-teller" bean.

There are two types of session beans:

- stateful – the bean contains non-persistent "conversational" information about a client, known as the conversational state of the bean
- stateless – the bean does not contain any client-specific data

Session beans and relational databases

Although session beans do not represent persistent objects, they often need to access persistent data. Developers using session beans may work with Java business objects that are stored in relational databases. The EJB specification does not address the issue of how to manage the underlying Java objects that represent the persistent state of a session EJB application. TOPLink addresses this problem by allowing developers to map the Java business objects used by the session beans to a relational database.

Session beans and TOPLink for WebLogic

Developers building session beans can access and store Java business objects that have been mapped using TOPLink. Both stateful and stateless session beans are supported. Any session bean deployed with the BEA WebLogic server can seamlessly access all of TOPLink's functionality. As a result, any existing object model that uses TOPLink can be used "as is" to create session EJBs without changing the TOPLink descriptors.

Session beans can be "TOPLink enabled" by providing bean instances with access to a TOPLink `Session`, which maintains all mapping information for a given TOPLink project and provides an interface for carrying out complex queries. Object-level transactions can be used with session beans through TOPLink's unit of work, which is integrated with the WebLogic Java Transaction Service (JTS).

In an architecture using session beans, TOPLink's full functionality and API can be used directly. The Java business objects used within the session bean can be read and stored in a relational database, through a TOPLink session or unit of work. The key coordinated aspect with the BEA WebLogic application server in this architecture is the integration of TOPLink with the server's JTS.

Entity beans

Entity beans represent a "business entity"; that is, a persistent data object. Entity beans may be shared by many users, are long-lived, and the data within the bean is expected to survive server crashes. Examples of entity beans include "sale item" beans, "bank account" beans, and "employee" beans.

The fundamental difference between entity beans and session beans is that entity beans are shared and persistent. The EJB specification provides an API and framework for EJB persistence, and presents the bean developer with two options: bean-managed and container-managed persistence.

Entity beans and relational databases

EJB persistence is more "generic" than the persistence available in relational databases. Persistence in EJB is defined more simply to allow for different kinds of persistence. As a result, many of the standard features of relational databases, such as relationships between data entities and dynamic querying, are not automatically available with EJB. These features of relational databases, and more, become available to EJBs only with the addition of an advanced object/relational mapping tool such as TOPLink.

Entity beans and TOPLink for WebLogic

The EJB specification offers bean developers two options for making their entity beans persistent: bean-managed persistence and container-managed persistence. These options are described in detail below.

Bean-managed persistence

In bean-managed persistence, the developer is fully responsible for making the beans persistent. This requires the bean developer to write persistence code within the bean itself, using direct file writes, JDBC calls, or a persistence framework such as TOPLink. The bean-managed persistence framework is particularly well suited for applications with simple entity bean models.

Bean-managed persistence and relational databases

Bean-managed persistence gives the bean developer control over how persistence is implemented within the beans that they build, and allows them to take advantage of any special features within the database on which their beans are deployed. This approach may improve efficiency in some cases. However, if the bean-managed persistence code does make use of database-specific information, the bean cannot be readily used with different schema or database types.

To preserve schema independence, developers can make calls from their beans to a persistence framework such as TOPLink.

Bean-managed persistence and TOPLink for WebLogic

The bean-managed persistence model requires the bean developer to write code for the EJB-defined callback methods such as `ejbFindByPrimaryKey`, `ejbLoad`, `ejbStore`, and `ejbRemove`. The developer can use TOPLink's API in these methods to work at the object/bean level instead of writing SQL or JDBC calls.

In most cases, if developers choose to use bean-managed persistence, they must write bean-specific persistence code within their domain classes. However, because TOPLink's architecture uses meta-data that generates database calls at run-time, many of these methods can be written in a generic way.

TOPLink provides classes and interfaces that make building bean-managed entity beans easier. Developers are given a high degree of flexibility, as they are not forced to use all of these classes and interfaces.

To optimize deployment, a shared TOPLink session should be accessible by many beans. This can be accomplished using the JNDI functionality provided by WebLogic.

Using TOPLink for WebLogic means that you can:

- define complex mappings between entity beans
    - one-to-one (1-1) and one-to-many (1-m) mappings between entity beans and regular Java objects
    - many-many (m-m) mappings between entity beans
- define complex mappings from entity beans to their attributes
    - direct collection mappings (map a collection of attributes to separate tables)
    - object type mappings (transform primitive Java types to other representations in the database)
    - serialized mappings (includes support for BLOBs and CLOBs)
    - aggregate mappings (multiple objects to the same table)
- easily map entity beans to multiple database tables (one class may be stored on several tables)
- define queries at the object level that can be used to write finder methods using TOPLink's advanced querying mechanism

It must be noted that the nature of the EJB bean-managed persistence architecture places some limitations on the persistence mechanisms. Some TOPLink performance optimizations and the unit of work feature are unavailable and under some circumstances bean identity can be lost. To leverage the benefits of these features with entity beans, container managed persistence is required.

Container-managed persistence

Container-managed persistence provides “automatic persistence” to the bean developer as well as the application assembler. The developer does not need to write any persistence code. Instead, persistence is handled based on information supplied in the bean’s “deployment descriptor”. In this case, the tools provided by the EJB container should allow the bean developer to specify complex mapping types for individual beans, and allow the developer to specify relationships between the beans that they have developed.

Container-managed persistence and TOPLink for WebLogic

With container-managed persistence, all persistence code is generated at deployment time by the WebLogic server and TOPLink.

TOPLink for WebLogic integrates directly with the WebLogic server to provide a container that automatically handles persistence to relational databases. Using TOPLink for WebLogic’s container-managed persistence capabilities, the bean developer can use TOPLink to specify the mappings between beans and relational tables, while the WebLogic EJB compiler generates all persistence code.

A developer uses the TOPLink Builder tool to define a TOPLink descriptor for the entity bean, which is then included with the bean’s own EJB deployment descriptor. This descriptor is used by the TOPLink container at run-time to handle all of the persistence requirements of the bean.

TOPLink for WebLogic allows bean developers to define relationships between beans, without forcing them to encode database information within their domain classes. The referenced objects are accessed simply through methods and the developer does not deal with database-specific details such as foreign and primary keys.

Clients can use TOPLink's rich querying and expression framework. TOPLink allows queries to be either stored or generated dynamically. The expression framework is object-oriented, allowing queries to be built using object attribute names rather than database field names, and provides a querying mechanism that reflects the object model rather than the underlying database implementation. This allows queries to be independent of the database. These queries can be used as the basis of finder methods or even invoked dynamically.

Developers are able to use high-level tools to define their persistence requirements and are not required to write any persistence code. All of TOPLink's mapping facilities are available to bean developers who use this option. As with bean-managed persistence, this includes the ability to:

- define complex mappings between entity beans
    - one-to-one (1-1) and one-to-many (1-m) mappings between entity beans and regular Java objects
    - many-many (m-m) mappings between entity beans
- define complex mappings from entity beans to their attributes
    - direct collection mappings (map a collection of attributes to separate tables)
    - object type mappings (transform primitive Java types to other representations in the database)
    - serialized mappings (includes support for BLOBs and CLOBs)
    - aggregate mappings (multiple objects to the same table)
- easily map entity beans to multiple database tables (one class may be stored on several tables)
- define queries at the object level that can be used to write finder methods using TOPLink's advanced querying mechanism

In addition to these mapping features, TOPLink for WebLogic's container-managed persistence solution also allows advanced features to be used, such as:

- transparent units of work (object level transactions)
- advanced object level querying
- object / bean identity
- caching
- batch reading
- batch writing

Summary

Enterprise JavaBeans (EJB) provide a framework that allows developers to build reusable server-side components for enterprise applications in Java. WebLogic is an industry-leading EJB application server. TOPLink for WebLogic integrates directly with BEA's WebLogic Server to allow developers to build EJBs using TOPLink's advanced object/relational mapping capabilities.

TOPLink provides EJB developers additional value above and beyond the EJB specification. These features include complex mapping, relationships between beans, advanced querying capabilities, and performance optimization features such as batch writing, batch reading, and object faulting ("just in time" reading).

TOPLink for WebLogic provides advanced functionality for developers who are building sophisticated applications using EJB and relational database technology. TOPLink allows developers to go further than the EJB specification for complex applications requiring mature object-relational technology.

The Object People and BEA have integrated TOPLink's object/relational technology and BEA's WebLogic Server to provide an industry-leading best-of-breed solution.

Additional information

The Enterprise JavaBeans specification is the main source of information about EJB. This specification is available on Sun Microsystem's website, http://java.sun.com/products/ejb.

Information about the BEA WebLogic application server can be found on BEA's website, http://weblogic.beasys.com.

The TOPLink family of products, including TOPLink for WebLogic, and information about The Object People's education and consulting services can be found at our website: http://www.objectpeople.com, by e-mailing info@objectpeople.com, or by contacting our sales office nearest to you.

Notes