- Search Resources
› Bugs
- Support & Training
- Technical Articles
- Newsgroups
- Documentation
- Downloads
› Partner Programs
- Marketplace
- Contribute
› Registration
› Site Feedback

Related:
- Early Access Program
- Module Proposals
- Partners Program
- Product Description

See Also:
- NetBeans[tm] Open Source
- Java Developer Connection[sm]
- Swing Connection
- Sun Developer Connection[sm]
- Dot-Com Builder

E-mail This Article
Printer Version

DEVELOPER RESOURCES
Technical Articles

Thought Inc. CocoBase

Dynamic Transparent Persistence over Local and Distributed Environments

CocoBase Enterprise Object to Relational, is an enterprise level, component style, mapping infrastructure, for local and distributed environments, developed by Thought Inc. to fully integrate with Forte[tm] for Java[tm]. Its Dynamic Transparent Persistence is fully portable, extensible, and scalable. This article discusses some features and benefits, and provides a simple introduction to the use of CocoBase as a development tool.

RELATED INFORMATION

Thought Inc.
Thought Inc. is the leading provider of O/R mapping, optimized for EJB and Java application development and deployment.

CocoBase Enterprise O/R
Product information and download available from Thought Inc.

Introduction

Thought Inc.'s product, CocoBase Enterprise Object to Relational (O/R) Dynamic Mapping for the Enterprise, is tightly integrated and optimized to work with Sun's Forte[tm] for Java[tm] IDE and iPlanet[tm] Application Server, as well as any standard J2EE[tm] application server. CocoBase enables developers to map data objects from a UML or XMI Modeling Tool (such as Rational Rose, for example), map data to and from database and objects, and generate Container Managed Persistence (CMP) and Bean Managed Persistence (BMP) Entity Beans, Session Beans, JavaServer Pages[tm] (JSP[tm]), Java classes, and transparent persistent Java and Enterprise JavaBeans[tm] (EJB[tm]). CocoBase, and its Dynamic Transparent Persistence, perform over local and distributed environments.

CocoBase simplifies database mapping for objects, enables code generation and its maintenance, facilitates tool/technology integration and the flexibility to evolve with technological changes. CocoBase is developed around the following core values:

Performance

> CocoBase's efficient design provides low overhead. The distributed caching model provides a unique O/R solution which allows a shared distributed cache across EJB servers and different component types.

Scalability

> CocoBase dramatically reduces the number of calls to the database – increasing user access, and decreasing resource usage.

Ease of Use

CocoBase's easy-to-use point and click mapping interface, and editable templates, facilitate custom code generation of Java code, CMP and BMP Entity Beans, JavaServer Pages, and Session Beans.

Tool and Technology Integrations

CocoBase runs as a module within Forte for Java, and is fully integrated with J2EE Application Servers, JDBC[tm] Drivers for Relational Databases, and UML Modeling Tools.

Flexible Architecture

CocoBase was designed from the outset as an enterprise level, component style, mapping infrastructure for local and distributed environments, with the ability to integrate new evolutions in technology.

Overview of CocoBase's Dynamic Transparent Persistence

Dynamic Transparent Persistence is vendor neutral and 100% portable across J2EE/J2SE compliant systems. There is no need for the developer to write database specific code in order to persist objects. Dynamic Transparent Persistence makes it possible to track and persist any object type, including EJB Entity Beans, CORBA objects, RMI objects, and Java classes — and without the requirements of query language in application code, or object model intrusion.

Dynamic Transparent Persistence allows the developer to persist relational data with a simple, non-invasive, widely distributable object. These capabilities operate as an extensible persistence layer – invisible to the object. Without any byte code or object model intrusion, CocoBase's dynamic transparent persistence runs on local, as well as distributed environments, and is designed to minimize maintenance without sacrificing performance.

CocoBase's Dynamic Transparent Persistence has been built for both J2SE and J2EE performance. Objects are versatile enough to be reusable, and to persist and manage objects simultaneously across a wide number of databases. Dynamic Transparent Persistence employs an independent persistent monitor – invisible to the application object – which persists, retrieves, and reconciles copies of objects to ensure proper network programming and serialization with its persistence facilities. Dynamic Transparent Persistence can also manipulate multiple copies within the same JVM[tm], so as to enable threaded servers to have a unique context and persist on parallel transactions.

CocoBase's stable object components are designed to allow developers to edit O/R maps without the need to change, re-compile, or re-process the object model. CocoBase's Dynamic Transparent Persistent object is unaware of its own persist ability: that is, the object is unaware of the data structure used to persist it, and is capable of being loaded or persisted to an arbitrary number of database tables. CocoBase works through its runtime facility to operate without awareness of the object. Lack of binding to a specific database design, enables objects to be more reusable: for example, CocoBase allows the developer to use the same map as an EJB in a J2EE application, or a Java Class in a J2SE application.

Integration with Forte for Java

A simple Java script installs the CocoBase module, which is then accessible from the IDE's Tools menu. This integration provides seamless access to the features and functionality of CocoBase Enterprise O/R.

Implementation of Dynamic Transparent Persistence

Maps and links can be created using a UML/XMI Modeling tool, or from reverse engineered object models, database schemas, or by hand. CocoBase runtime classes can then be used to provide transparent persistence to a Java Object Model which corresponds to the created maps and links. Again, CocoBase accomplishes transparent persistence with Java Object Models without using byte code manipulation, proprietary interfaces, or class hierarchy intrusion. As a result, no special classes or interfaces are required in the Java Object Model in order to achieve persistence with CocoBase — the only requirement being that the Java Object Model must have a default constructor with no arguments (which is also standard Java coding practice).

There are three basic runtime components involved in the dynamic transparent persistence of objects:

- The CocoBase Runtime O/R mapping class. This class wrappers the JDBC driver, issues queries, and makes the actual persistence calls. For example, `thought.CocoBase.CocoPowder` or `thought.CocoBase.CocoPowderPlugin20` (for JDBC 2.0 connections).
- The `thought.CocoBase.Transaction` object. This object tracks changes of instances, and acts as a change 'buffer'. If a Transaction object is used, it will only call a CocoBase runtime driver (an O/R mapping runtime class) when `txn.commit()` is called — thus reducing database traffic and overhead.
- The `thought.CocoBase.navapi.Navigator` object. This object tracks and detects changes in relationships based on link definitions.

The `Navigator` class can operate either in conjunction with a `Transaction` object, or standalone. While complex object graphs can be transparently managed directly by the `Navigator` object — and without using the `Transaction` object — use of the `Transaction` object is generally preferred because of its buffering and update optimizations, which only persist those attributes which have changed.

Developing Applications with CocoBase's Dynamic Transparent Persistence

Having properly created the links and maps, corresponding Java classes can be generated by choosing **File** > **Generate Java Code** from the CocoBase menu bar. The code generation target, `Default CocoNavigate Java Transparent Object`, will generate a pure Java class with no CocoBase interfaces. Once these generated classes have been compiled, CocoBase runtime classes may be used to persist instances of them.

First, a CocoBase connection should be opened. The following code illustrates how to accomplish this.

```
CocoDriverInterface myBase =
CocoDriver.getCocoDriver(
   "thought.CocoBase.CocoPowder",
   "org.hsql.jdbcDriver",
   "jdbc:HypersonicSQL:hsql://localhost;cocoprop=
       cocofactory=CocoProxyFactory",
   "sa", ""
)
if(myBase.connect() == -1) {
   System.out.println("Failed connect!");
   System.exit(1);
}
...
```

Next, a CocoBase transaction object is created to manage any objects which are retrieved. Notice in the following code, how the transaction object is configured through parameters and property settings.

```
thought.CocoBase.Transaction cocoTxn =
      new thought.CocoBase.Transaction(myBase,
false);
Properties cocoTransProps = new Properties();
cocoTransProps.put("preserveCommit","true");
cocoTransProps.put("commitconnection","true");
cocoTransProps.put("throwExceptions","true");
cocoTransProps.put("updateOnlyChangedColumns","true");
cocoTxn.setProperties(cocoTransProps);

// Begin a new transaction.
cocoTxn.begin();
...
```

The following code illustrates how to open a CocoBase `Navigator` object for the Navigation model being used, and also how to register the `Transaction` object with the Navigation model.

```
// Instantiate a Navigator with the Link model
created from
// the UML/XMI document
thought.CocoBase.navapi.Navigator navigator =
    new thought.CocoBase.navapi.Navigator
(myBase, "company");
// Assign the current transaction to the
Navigator
navigator.setTransaction(cocoTxn);
```

In the following code example, the `CocoProxyM` class ‘wrappers’ the pure Java

Object Model class, and provides compatibility with CocoBase through Java reflection — that is, instead of requiring any special interfaces in the Java class itself. The top-level node should be selected in order to access this area of the code.

```
// Setup our object to query
Department dept = new Department();
department.setName("SALES");
// This will read & bind all 'Department' objects
to the transaction.
Vector deptVector = myBase.selectAll(
  new
thought.CocoBase.CocoProxyM(dept),"Department");
```

With CocoBase's Dynamic Transparent Persistence it is possible to step through each of the objects and identify the retrieved object to be navigated using the `loadAllLinks()` method, which does the automatic navigation for that object:

```
for(int i=0; i< deptVector.size(); i++) {
   Department d =
(Department)deptVector.elementAt(i);
   // Because the cascadeLoad flag is set to true
in the
   // direction Department->Employees, the
employees link
   // will load automatically
   d = navigator.loadAllLinks(d,"Department");
   ...
   Vector emps = d.getEmployees();
   for (int j=0; j<emps.size(); j++) {
   // raise salaries by 20%
   emp.setSalary(emp.getSalary()*1.2);
   ...
   }
   ...
// Once changes are made to an object graph those
changes can be
// synchronized using the updateAllLinks method
such as:


navigator.updateAllLinks(d,"Department",true);
}
```

Committing the buffered object changes is undertaken with the following method call:

```
cocoTxn.commit();
```

The preceding code introductions are fairly small, and can be run entirely on the server side using Entity or Session beans in J2EE environments without byte code or object model intrusion. Regarding locally executed non-J2EE applications, the application intrusion is very small – with only the 'load', 'synchronize', and 'persist' operations

requiring a single method call for each root node.

The `Navigator` object supports one-to-one, one-to-many, and many-to-many link definitions with cycle detection. It can also detects these locally or by reconciling serialized or copied objects, again without byte code or object model intrusion. This technology is true transparent persistence, architected and designed for the JVM[tm]-oriented Java language.

Conclusion

CocoBase Enterprise O/R, Dynamic Mapping for the Enterprise, has been developed to fully integrate with Forte for Java. The latest version includes powerful and versatile Dynamic Transparent Persistence functionality. Developed around a set of core values, CocoBase's Dynamic Transparent Persistence is fully portable, extensible, and scalable. Among its many features are the ability to perform true transparent persistence in both local and distributed environments.

READER SURVEY

I found this article...

Not Informative **Informative** **Very Informative**

Comments:

Submit

This page was last modified: 06.Nov.01

Developer Resources : Search Resources | Bugs | Support & Training | Technical Articles | Newsgroups | Documentation | Downloads | Marketplace | Contribute | Partner Programs | Site Feedback

Copyright 1994-2000 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 USA. All rights reserved.
Terms of Use. Privacy Policy. Feedback