



**Product
Information**

*Enterprise Features & Requirements
Analysis For EJB3 JPA & POJO
Persistence*

CocoBase® PURE POJO™ Uniquely Provides BEST IN INDUSTRY Support For The Full Range Of Enterprise Features And Requirements For Corporate Applications.

CocoBase® PURE POJO - Technical Paper

INTRODUCTION

CocoBase® Pure POJO™ V5 Distinguishes Itself As The Best Choice ORM Tool For Building Enterprise Applications Due To Its' Unique Ability To Deliver The Full Range of Simple to Complex POJO Persistence Functionality and Performance For Persisting Data.

The patented CocoBase Mapping Layer architecture is unique among ORM tools and provides the only solution that provides the full range of simple to complex functionality required for building enterprise applications. Most ORM tools only support "desktop simple" and "workgroup moderate" level functionality with minimal support at the "internal enterprise" level and essentially none at the advanced "external enterprise" level of functionality.

This is highly problematic and can create the situation where at some point in the development cycle the developers will have to create additional features and work arounds themselves, if possible, or be required to find a different ORM tool just to be able to finish the application. Either result greatly reduces productivity and increases project costs which needs to be avoided. The CocoBase architecture was focused on solving enterprise requirements from its' beginning and provides an excellent solution for enterprise development.

CocoBase® Architectural Overview

CocoBase provides a powerful yet simple to use patented Dynamic Mapping Layer architecture for persisting POJOs in enterprise applications. This approach decouples POJOs from the database representation (i.e. tables, fields, etc.). This database information is then stored in a dynamic and highly flexible CocoBase map in a repository outside of the application.

The Dynamic Mapping Layer architecture removes database specific code and predefined SQL from the POJO and instead, relies on user defined database maps to generate application specific SQL and JDBC code at runtime. SQL instructions are by default dynamically created and bound to object fields at runtime based on the mapping definitions. CocoBase also allows developers to pre-compile, tune and map such SQL instructions to and from object fields at design time as well, independently from the application code, for each of the CRUD (Create, Read, Update, Delete) database operations.

The Dynamic Mapping Layer approach also provides the flexibility to quickly implement custom caching, custom routing and other custom behaviors that are typically needed for enterprise-level applications. These custom behaviors do not require recompiling of the application. For example, if you write your application against an Oracle system, and find that your version of Oracle and your platform doesn't perform fast enough, you can install a plug-in in-memory cache database system for your applications on the fly without changing any of the object definitions and no need to recompile the application. This provides an enormous time savings as well as an amazing amount of flexibility that keeps the development process simple and easy to deal with.

The CocoBase Dynamic Mapping Layer persistence architecture also allows for attributes to be mapped to single or multiple database tables simultaneously, in both local and distributed environments. The same object model can be readily used in different applications, each specifying unique, user-defined persistence, SQL and navigation models. This ability to standardize on one class model and/or one set of maps for a set of applications greatly reduces project costs and extends the savings to multiple applications with little to no additional work.

A key part of the CocoBase architecture is the Persistence Manager which comprises a set of easy-to-use and comprehensive programming APIs. It implements the POJO persistence model and is responsible for automatically and transparently managing the state of persistent objects. The CocoBase Persistence Manager is known for its incredible performance and scalability, which more precisely means that the execution time required for persisting a complex graph of objects, grows linearly to the number of objects in the graph. No other ORM tool available has been shown to have linear performance and scalability characteristics.

Key Architectural Benefits

- 1) **Extremely powerful persistence architecture** based on the Dynamic Mapping Layer approach that uniquely supports the full range of EJB 3.0 and POJO persistence requirements from simple to the most advanced as no other ORM tool is able to do.
- 2) **Extreme ease of use and high productivity with all new CocoBase GUI's;** including the Mapping WorkBench and Magic Mapper that intelligently auto-maps the tables, fields and objects together for the developer. No other ORM tool has such a complete GUI toolset for its' development. The GUI's give the developer a simple step by step approach for dealing with even the most complex persistence requirements while at the same time making available hooks into the tooling that allows total access to all functionality with the ability to modify it.
- 3) **Extreme high performance** (can be up to 3 times faster than JDBC) with extensive methods for additional performance enhancement including caching.
- 4) **Extreme high volume scalability with unique linear performance** characteristics that is designed to handle applications with a large degree of transactions and users typically found in enterprise-level applications.
- 5) **Full breadth of querying solutions** including EJBQL (a pure Object-Centric Querying syntax), high level CocoBase® Query API that looks very similar to SQL, low level CocoBase® Query API, standard CocoBase® Query by Example which queries instances instead of object models.
- 6) **Architectural extensibility that supports integrations with Eclipse IDE, Spring** Application Framework, Application Servers, Relational Databases, UML Object Modeling tools, etc.

CocoBase® Features List Organized From Simple To Advanced

Level One - “Desktop Simple”

Includes prototypes, desktop applications, small volume of both data and users, not performance sensitive, with simple modeling and database requirements.

Level Two - “Workgroup Moderate”

An increase from level one with some complexity in mapping and data volume that is generally referred to as a departmental level application.

Level Three – “Internal Enterprise”

An increase from level two with the additional focus on Infrastructure and Corporate Applications with complexity, performance and reuse requirements involving multiple modules and with either high volume of users and/or high data volumes within critical parts of the system.

Level Four – “External Enterprise”

An increase from level three with the additional focus on Internet Deployed Applications, Legacy Data Model integrations including coexistence with poorly defined legacy database models, high data and user volume, complex requirements in modeling and critical path enterprise applications where response time and custom behaviors are critical to business competitiveness.

<i>OR Mapping Features</i>	Level One	Level Two	Level Three	Level Four
<i>Direct Mappings</i>				
• Basic Attribute Mapping	x			
• Composite Attribute Mapping (unlimited nesting levels)	x			
• Flexible Table Join Mapping		x		
• Virtual Attribute Mapping*		x		
• Attribute Mapping Override (in subclass types)			x	
• Read-Only or Write-Only Column Mapping			x	
• Derived Attribute Mapping (calculated from db column values)				x
<i>Object Relationships</i>				
• One-to-One Relationship Mapping	x			
• One-to-Many Relationship Mapping	x			
• Many-to-One Relationship Mapping	x			
• Many-to-Many Relationship Mapping	x			
• Inverse Relationship Mapping (i.e. bidirectional relationships)	x			
• Circular Relationships	x			
• Transparent Foreign Key Mapping and Management	x			
• Ordering Support (for persisting ordered collections, e.g. java.util.List)		x		
• Map Key Support (for java.util.Map based relationships)		x		

<i>OR Mapping Features</i>	Level One	Level Two	Level Three	Level Four
• Lazy/Eager Relationship Load Modes		x		
• Operation Cascading Control (independent per operation)		x		
• Relationship Mapping Override			x	
• Relationship Qualifier Expressions (for relationship filtering/ordering)			x	
• SQL Cascading And Filtering Control				x
• Virtual Relationship Mapping**				x
Type Inheritance				
• Single Table Inheritance Mapping	x			
• Joined Table Inheritance Mapping	x			
• Single Discriminator Column Mapping	x			
• Separate Table Inheritance Mapping		x		
• Multiple Discriminator Column Mapping		x		
• Combined Inheritance Mappings		x		
• Abstract Superclass Or Interface Mapping			x	
• Multiple Inheritance Mapping				x
• Discriminator Mapping Override				x
Object Identity and Versioning				
• Single Primary Key Mapping	x			
• Composite Primary Key Mapping (primary key class not required)		x		
• Sequence Generator Mapping		x		
• Table Generator Mapping		x		
• Identity Column Generator Mapping		x		
• Versioning Column Mapping		x		
• Custom Generator Mapping			x	
• Optimistic Locking Modes (independent for each attribute)			x	
• Virtual Primary Key Mapping* (i.e. transparent object identity)				x
Querying & Advanced SQL				
• Named Query Mapping (EJBQL and Mapping Based)			x	
• Customizable SQL Mapping (independent for each SQL CRUD operation)				x
• Stored Procedure Mapping				x
• SQL Realms (for SQL re-routing)				x
• Native SQL Query Mapping				x
• Classless Mapping (no Java business class required)				x

*allows a column to be mapped even without a corresponding field in the object class (useful for querying and for mapping overrides).

** allows links between objects to be maintained even when there's no corresponding reference fields in the related object classes (useful for operation cascading control).

<i>Runtime API Features</i>	Level One	Level Two	Level Three	Level Four
High-Level Easy-to-Use Persistence Manager API	x			
Optimistic Locking Support	x			
Pessimistic Locking Support		x		
Lazy Load (for relationship fields)		x		
Independent Fine Grain Cascading Control for each operation/relationship ***			x	
Custom Instance Life-Cycle Callback Listeners			x	
Custom Instance Factories			x	
Custom Member Accessors/Mutators (for non-POJO or reflection based access)			x	
Custom Caching Implementations			x	
Smart Batch Updates, Inserts and Deletes			x	
Statement Caching			x	
Distributed (e.g. 3-tier) Lazy Load				x
Custom Relationship Proxies				x
Core Runtime Access Plugins (for security, filtering, pre/post processing)				x
Notification Across Multiple Persistence Manager Instances				x
SQL Re-routing through SQL realms				x

*** cascading operations include *insert, update, delete, merge* and *refresh*

<i>Dynamic Querying Features</i>	Level One	Level Two	Level Three	Level Four
Query-by-Example (for attribute-based querying)	x			
Local Cursor Based Querying		x		
EJBQL Object Based Querying		x		
API Based Querying			x	
Raw SQL Dynamic Mapping and Querying				x
Distributed Cursor Based Querying				x
Combined Querying Strategies				x

<i>Mapping Repository Features</i>	Level One	Level Two	Level Three	Level Four
Highly Efficient Thread-Safe Access to Mapping Repository Metadata	x			
Repository Metadata Caching		x		
Extensible Repository Framework****			x	
Runtime Repository APIs (repositories can be created/managed at runtime)				x

**** allows mapping metadata to be stored in virtually any format (xml, binary files, database tables, etc.)

<i>Workbench GUI and Tools</i>
Extremely Easy-to-Use High Productivity Mapping Workbench GUI
Java Class Importer
Database Table Importer
Magic Mapper that intelligently auto-maps Java classes to database tables
All-In-One Project Wizard
Mapping Validation integrated into GUI mapping process, including detection and description of common mapping errors

<i>Common Enterprise Application Requirements Supported by CocoBase®</i>
Legacy and Unstructured Database Models supported without requiring them to be changed
Legacy Coexistence Supported, while permitting new applications to define Structured Models from Unstructured Data
Database Schema Integration
Distributed Caching and Transactions
Data Transformation and Filtering
Auditing and Security
Mapping Validation facilities provided in the Runtime
Extensive and fine grained extensible debug logging messaging supported
EJBQL Object Oriented querying based on support for Table Spanning maps, not constrained by Legacy Database Models
Virtual Relationship Constraints handled at the Persistence Manager layer enforced
Temporal Database Models Supported
Central Shared Mapping Repositories that support map reuse across projects, and multiple projects within a repository
Version Control System Integration, including within Eclipse
Integration with Other Enterprise Tools, particularly Spring, Eclipse and an extensible architecture for the future
Extensible Persistence Frameworks
Multiple Database Locking Strategies Combined
J2EE/JPA Integration while supporting advanced mapping facilities not expressed in the EJB specifications
High Productivity of both Development and Maintenance process
High Manageability of Mapping and Modeling
High Performance that remains linear
High Scalability in terms of Model complexity, Data volumes and number of Users

Conclusion - Be Sure To Use An ORM Tool That Can Handle ALL Of Your Applications' Requirements.

The key challenge when choosing an ORM tool for a project is whether or not it can handle the levels of complexity of the application being developed. This is especially important for enterprise applications because if any requirement is not supported then the whole project can fail to deliver within schedule and budget constraints, or at all. To avoid this situation, the ORM tool must be carefully assessed which is facilitated by understanding its' persistence architecture. The architecture is the foundation of the ORM tool and will define what can be developed with it.

For example, an ORM tool with a simple architecture will be only be successful for easy desk top applications that support few users and have limited technical needs. If additional features or requirements will be needed later, then the ORM tool or platform will probably have to be replaced or upgraded. To meet these expanded needs, the application will need to be re-coded to use another ORM tool or migrated to hardware and software platforms with more capacity. This can be a rather expensive and time consuming task that will most likely cause the development schedule to fall behind the competitive needs of the company.

Once the various ORM architectures are understood, then a detailed look at what levels of complexity can be supported is the next step. Most ORM tools are only designed for desktop applications with simple to limited medium level persistence requirements. They do not have the ability to manage the range of needs required to successfully deliver in the enterprise environment. CocoBase® PURE POJO™ was architected from the beginning to cover the range of requirements found uniquely at the enterprise level of requirements. This means that CocoBase can be used for desktop as well as enterprise applications and can be expected to handle the full range of application requirements from simple to complex quite easily.

The key in this process is for the IT Manager to make sure that the decision process for selecting an ORM tool is based on a methodical and rational approach. The application requirements need to be carefully taken into account including all of the current and expected needs that the company has. Any deviation from this approach can allow bias, inexperience, impatience and other counterproductive influences the choice of an ORM tool that will not adequately deliver success from the beginning of the project to the end.

Company Information

THOUGHT Inc.®, in business since 1993, has been shipping the CocoBase® Enterprise O/R product since early 1997 and is currently in its' 5th major release. THOUGHT Inc.® invented and patented repository based Dynamic Object to Relational Mapping™. CocoBase® is by far, the most mature Java based O/R Mapping tool available and leads the industry in technological innovation. This is why so many of our customers rely on CocoBase®!

For more information please see the website at www.thoughtinc.com.

LEGAL NOTICES

Copyrights

Copyright 2007, THOUGHT Inc., 5 Third Street suite 815, San Francisco, CA 94103 USA. All rights reserved. Copyright in these (any and all contained herein) documents is owned by THOUGHT Inc. This material is for personal use only. Republication and re-dissemination, including posting to news groups, is expressly prohibited without the prior written consent of THOUGHT Inc. This publication is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement, to also include any and all technical inaccuracies or typographical errors.

Patents

CocoBase® is a patented product under patent #5857197 as well as patents pending for object caching, object navigation, dynamic object querying technologies, etc.

Trademarks

CocoBase®, THOUGHT Inc.®, Dynamic O/R Mapping™, Dynamic Transparent Persistence™, Pure POJO™ and others are all trademarks of THOUGHT Inc.® All other trademarks are owned by their respective owners.