THOUGHT INC.

The Dynamic O/R Mapping Company™

CocoBase® Technical Paper

Guidelines For Successfully Using Enterprise JavaBeans With CocoBase® Enterprise O/R

J2EE TECHNOLOGY

GUIDELINES FOR SUCCESSFULLY USING ENTERPRISE JAVABEANS WITH COCOBASE® ENTERPRISE O/R

CocoBase® Technical Paper Expanding on the WebSphere Advisor Article "The Truth About EJBs"

John Kidd, from Kidd and Associates, wrote a very helpful article for the WebSphere Advisor delving into the issue of successfully using Enterprise JavaBeans. He exposed the truth behind the common myths of using EJBs and guides the developer as to how and when they should be used. We, here at Thought Inc.®, agree with a lot of the points in the article and have included below additional guidance on using EJBs and specifically how CocoBase® Enterprise O/R can either help the developer make the most of EJBs or resolve some of the design dilemmas that EJBs introduce.

The format of this tech paper brings forth advice and observations from the article, with elaborations on how developers can use CocoBase® to successfully deal with or prevent the particular issue. The developer can use this paper to further clarify when using EJBs is the optimum approach and provide some guidance as to how to accomplish a design goal using the most appropriate EJB technology and designs. Please note that the technical support team at Thought Inc.® is always available to provide further details and guidance to developers regarding using EJBs with CocoBase®.

Please see the original article at http://doc.advisor.com/doc/12808

"MYTH ONE – EJBS ARE TOO SLOW"

Performance problems are an all too common issue and many of the causes are covered in-depth in the WebSphere article. For developers using WebSphere and EJBs, they can follow the advice in the article by simply using the CocoBase® Object to Relational Mapping tool to automatically solve many of the issues brought forward in the article.

"Performance"

So lets begin with the article's observation about performance; "For every

update method for a piece of information in the entity, an EJBLoad/EJBStore pair executes…"

This is definitely the default behavior of Entity Bean EJBs. This behavior can be improved by a well-written Entity Bean (such as one generated using CocoBase®), which will prevent these secondary loads and saves in most cases. The CocoBase® code generation templates produce beans that more intelligently load beans - especially the first time they are accessed or if they are all accessed within a transaction. CocoBase® generated Entity Beans will auto-detect if any attributes have changed, and will only save beans that have changed.

CocoBase® generated beans also provide Value Objects that correspond to the data represented by the entity bean. CocoBase generates accessor methods on the Entity Beans which return and receive instances of these value objects. With these accessor methods, developers can use the Value Objects to conveniently access all of the attributes within the bean in a single Entity Bean method call. The individual attributes can then be extracted from the Value Object on the client greatly reducing network traffic and data access performance associated with complex Entity Bean data.

CocoBase® also supports 'dependent objects' for entity beans. For a sample scenario, we can assume a Customer Entity Bean, which has a relationship to Order objects with 1-m multiplicity. With CocoBase®, that relationship can be defined as a Standard Link (STD_LINK) type and the dependent objects are a serialized collection of value objects sharing the same ejbLoad, ejbStore, related transaction and the same JDBC connection pool. This can reduce overhead significantly for entity beans by reducing the amount of SQL, the number of JDBC connections and transaction contexts, the number of Entity Beans that must be pooled and managed by the server as well as the amount of network traffic necessary to access a more complex model of data.

"Listing behavior – Also Known as Cursor"

This is the common issue in EJB where there's no explicit cursor control. With Entity Beans when a query is issued, at a minimum, all primary keys are loaded even when the bean value is never accessed by the client. Because of this design, a Bean Remote reference is automatically created for each row of data

no matter how large the result set! For instance if a developer issues a query that returns 10,000 objects, there are 10,000 Entity Bean references created and 10,000 primary keys retrieved from the database. CocoBase® offers as an alternative, scrollable cursors, which are used with Session Bean based persistence. This will be covered in more detail further in this document and when dealing with large result sets this is an indispensable optimization and feature.

The CocoBase® approach is very consistent with other advice related to EJB usage. The scrollable cursor can be opened with CocoBase®, a block size for the cursor can be set and only those pages retrieved will be actually loaded into the server and client memory. This is a much more scalable and enterprise capable architecture - especially in environments where large data volumes will be paged through either partially or fully.

"Entity relationships"

The article says "...This leads to a chain reaction: You look up entity number one, which has a relationship with entity two that has to be loaded, which... and so on [...] Solution? Have a good reason for defining relationships in your entities. "

This is interesting advice from the article and it is consistent with almost any partially defined persistence architecture such as EJB 2.0 CMP. There are several additional problems with EJB 2.0 Entity Relationships that are not mentioned that the developer should be aware of. Most of the issues arrive from either inconsistently or incompletely defined aspects of the EJB persistence architecture, and they are typically easily overcome by a commercial O/R product with more defined specifications and implementations such as CocoBase.

One of the unmentioned related issues is that Entity Relationships are only supported with Local Interfaces. The conceptual intent of EJBs is founded in the concept of remotely accessible and highly distributable objects. EJBs are supposedly designed to be 'Enterprise' and 'Distributed' components, yet to obtain Entity Relationships it is not possible. The EJB 2.0 Entity Relationships are the complete opposite of this principle as they only run 'locally' and are only 'non-distributable'. Considering how much richer other 'local' persistence

architectures and implementations such as CocoBase® 'Local' Transparent persistence provide, it's really unclear why Local Interface Entity Beans even exist.

Another issue is that there are not explicit controls over the other cascade operations such as insert, update, delete in a fine grained and consistent manner. This means complex graph management is often incompletely or inconsistently implemented. A properly implemented persistence framework provides developers with these controls and in the case of CocoBase®, they are provided in both local and distributed environments.

And finally there is no way to define that a graph should be lazy loaded on access. This is why cascading is such as large issue with EJB 2.0 relationships, and why a more robust and complete persistence architecture such as CocoBase® is critical for environments that have inter-related data not well suited to the EJB 2.0 CMP architecture. It's important to note that CocoBase® does provide its own CMP implementation, but because of the CMP specification this is restricted in functionality somewhat by the design of CMP itself.

If distributed EJBs are needed, CocoBase® BMP Entity Beans can have remote interfaces and CocoBase® Distributed Dynamic Transparent Persistence further gives distributed lazy controls with Value Objects and no complex Entity Bean programming. Since the CocoBase® cascade controls are on a per link basis, it's perfectly safe to use object models as they really exist in the Java Language. By using CocoBase® you don't need to be afraid that expressing a business relationship in your component model will trigger a server overload on the application server because there isn't sufficient control to load the objects.

And finally, CocoBase® Distributed Dynamic Transparent Persistence provides the ability to implement this with scrollable cursors and even the ability to work 'offline' in a web application and re-establish a connection and set its context to provide data safe optimistic locking automatically!

"Misuse"

The article talks about "Start by using stateless beans. Only advance to entities and stateful beans if you require them".

This is one of the best and clearest pieces of advice both in the WebSphere Article and for EJB programming in general. It's advice that all EJB customers need to be aware of. Since Entity Bean EJBs generally reduce performance and scalability on truly enterprise applications, developers should be aware that by using CocoBase® to provide your WebSphere persistence this problem is greatly alleviated. CocoBase® can successfully manage a wide range of complex models as well as higher data volumes than Entity Beans can. This can be accomplished by using stateless or stateful architectures, or a combination of both which expands the choices for the developer.

"MYTH 2: EJBS AREN'T SCALABLE "

The article says "…Fortunately, we've moved beyond this myth. Today, there are numerous J2EE applications that meet huge service loads."

This is a very 'caveated' statement. While this is true there are numerous scalable J2EE applications, it's typically 'Session' Bean J2EE applications and not Entity Beans that meet this requirement. It is most certainly possible to produce a reasonably scalable application with Entity Beans, but only if the proper transactions are used, and it very much depends on the data volumes and relationships of data that are required by the application.

It's possible for developers to help solve some of the fundamental Entity Bean issues by using other coding techniques like Value Objects, and by restricting the application with very limited or controlled queries. The EJB Entity Bean implementations typically are much harder use when creating a high performance and highly scalable application in contrast to other approaches.

"MYTH 3: EJBS ARE DIFFICULT TO WRITE "

The article says "…However, there are now several good tools in the market that create EJBs for you. "

This is true and CocoBase® is a prime example of this. With CocoBase®, you can generate EJBs with features simply not possible in other tools; such as

including Entity Beans that can span multiple database tables, have their SQL tuned on the fly, issue dynamic EJBQL (with EJBQL 2.1 and extended Outer Join syntax), automatic remote interface to bean links, dependent object support and a variety of other features. The myth is based on the reality however that Entity Beans are inherently complex compared to standard Value Objects and Java Classes. Even a tool like CocoBase® doesn't fully reduce the complexity of the generated code, or the complexity of the Entity Bean architecture.

The article says "…Option 3: Use a stateful session bean. By far, this is the easiest approach to use. "

This is great advice. Please note that with CocoBase®, you can use a single stateful session bean to manage all of your business objects per session. This means that with 1000 simultaneous users, you still only have 1000 session beans in memory, which is not a very large number for a typically loaded application server. And with CocoBase® you can couple this with our stateless compatible handling of 'original state' with a single push() method call before saving your object model. This means that the number of live stateful sessions can often be further reduced to a tiny number while still retaining J2EE transaction handling.

"BOTTOM LINE ON EJBS"

This section of the article can be a little vague because it doesn't distinguish between Session and Entity Beans. So we decided to clarify where, in our experience, each statement holds true for Entity vs. Session usage.

The article says "…Choose EJBs when you know your application will scale beyond initial low usage levels. "

This can be very good advice unless you also need scrollable cursor services or lazy operations. Entity Beans sacrifice performance for being able to access large numbers of entity beans. The 'virtual memory' characteristics of the Bean Pooling allow for large numbers of objects to be loaded, but the price for this is a constant issuance of new SQL statements for each and every bean load and save which produces a very slow architecture when large numbers of

Entity Beans are used. Typically large usage levels perform significantly better with Session Beans than with Entity Beans, so this statement is suited to that clarification.

The article says "…Choose EJBs if you need transaction management. "

This is true for both Session and Entity beans. Both component types can issue JDBC and SQL within a transaction context, and both are useful in the case where 2-phase commit is required. It's important to be aware, however, that if a developer doesn't require 2-phase commit, the overhead of using any EJBs is unwarranted. A simple connection pool with an O/R layer such as CocoBase® can still issue JDBC commit and rollback behaviors without requiring any transaction coordinator. So this statement is true, only when you need transaction management to be 'implicit', or when there are multiple databases involved in the same transaction. The JDBC driver and the database manage transactions for a single connection quite well and require no coordination or special J2EE support.

The article says "…You don't need EJBs for read-only systems (in fact, don't use them!). "

This is true! You don't need them for write systems either when a single database or a heterogeneous database environment is accessed. You can get 2-phase commit transactions without using any Entity Beans (just Session beans) and in some cases you can even accomplish this with just JSPs and no session beans for the platforms that provide explicit access to UserTransaction objects and DataSources directly from JSP clients.

The article says "…Design: Use DAO classes for read and listing. Use EJBs only for update type operations. "

This is good advice, and even better advice is to use a 'facade' such as the CocoBase® Distributed Dynamic Transparent Persistence facade that provides a more 'generic' interface and doesn't require a DAO per object type. This CocoBase® design pattern tends to be easier to code, more sophisticated in the kinds of problems it can handle and also uses less server resources especially when combined with a stateful session bean architecture. Developers can treat

the façade implementation as a 'generic' DAO, and therefore reduce the amount of coding and complexity typically required in application specific DAOs.

Part of the complexity of Entity Beans is that they are in fact a distributed DAO, and DAOs wrappering these distributed DAOs doesn't simplify things, but in fact complicates J2EE coding. The façade approach is a better fundamental approach, much easier to use and much better at handling complex object models.

The article says "…Use EJBs if you have different technology-based clients talking to your business logic. There is a benefit to isolating business logic at the mid-tier."

Because Session Beans are meant to capture this sort of 'process' behavior, Entity Beans are still not required to accomplish this sometimes critical task. This server side logic should in fact be a Session Bean feature, and not an Entity feature typically.

The article says "…You're making a long-term commitment to J2EE. "

Even then, remember that you don't have to use ALL J2EE facilities just because they are available. Many times developers use technologies because they are available even when they aren't the most efficient or the best to solve the problems.

The complexity of Entity Beans is not offset by any features you obtain from them when compared to the capabilities provided by Session Beans coupled with a Commercial Object to Relational Mapping tool such as CocoBase®. Especially in light that many of the most advanced Entity Bean features (2.0 relationship management) don't even work in a distributed manner, making them useless for remote clients from other languages, wiping out arguably the best reason to use Entity Beans. Using Value Object Models and either a local or a distributed Session Bean architecture will generally be the best choice for new J2EE development.

Any developer considering Entity Beans should ask themselves, 'What it is giving them that isn't provided by the Session Bean architecture.' And they should also ask, 'What they would be giving up with Entity Beans that would be available when using a Session Bean & O/R Mapping approach?' Features such as Inheritance, explicit control over object model loading, an object model that is distributable, scrollable cursors, tunable SQL, re-use of model across J2EE and

J2SE platforms, extensible caching, advanced dynamic querying for reporting and other uses, and a variety of other features almost too numerous to include.

Because Entity Beans and Session Beans are standard J2EE component and programming APIs, using Session Beans alone still provides a J2EE architecture and yet, with the Session Bean method of coding, the developer retains much greater control over the explicit object types (with inheritance for instance) and a variety of other features not directly possible with Entity Beans.

We hope this technical article has been helpful and further supports your successful use of J2EE combined with CocoBase®. And you join the many companies who have successfully used CocoBase® for their corporate solution to using J2EE Enterprise JavaBeans.

If you have further technical questions or require any clarifications please email to support@thoughtinc.com.

If we can help you with your sales needs, please contact sales@thoughtinc.com and please remember that you can request and download a 30 day free evaluation of CocoBase® directly from our website at http://www.thoughtinc.com.

Company Information and History

THOUGHT Inc.® is focused on delivering Dynamic Object to Relational Mapping™ OPTIMIZED for EJB and Java. CocoBase Enterprise O/R is the key to success for companies managing information in Java applications for the J2EE and J2SE platforms. The CocoBase tool increases performance by up to and over 20,000%, decreases database access development / deployment / maintenance costs by up to 85% and works with any standard J2EE - J2SE Application Server, making it's use a key part of any companies tool set.

Technology: Dynamic O/R Mapping™ - dynamically maps and manages data in applications between the object and the relational database.

Platforms: J2EE, J2SE, J2ME

Implements: EJB CMP / BMP Entity Beans, EJB Session Beans, Java Server Pages, Dynamic Transparent Persistence™, Java persistence.

History: THOUGHT Inc.'s founders lauded the advent of Java as a new and welcome beginning in programming paradigms. With years and years of industry experience, we have crafted products that put the power of java, and all its potential in the hands of the developer. The beauty of CocoBase stems from the simplicity and yet the sophistication of its design.

The company started in the spring of 1993 focused on delivering Object Oriented Solutions to large corporate customers. Soon thereafter, the engineers at Sun Microsystems creating Java, persuaded the founders to look at this new programming language. The Java programming language provided a perfect platform to deliver mapping technology that had been worked on for years by the Thought Inc. founding technologist.

In early April of 1997, the first product "CocoBase Enterprise O/R" was delivered. Over the years different companies have licensed the technology such as Sun Microsystems. Currently, over a 100,000 copies of CocoBase have shipped making it one of the most successful Java tools in the industry.