Data Object Transactions

Transaction boundaries and dependencies

Marcel Baumann

Version 0.1.2

Table of Contents

1 Introduction

The data object framework *DOG* is used in a mission critical application for contract management in the insurance branch. The principles of this framework are discussed in [1].

The development team learned a lot during the migration from CORBA based servers to J2EE container. The container owns the transactional context and the persistence layer – based on OJB – must interact with it. During the transition the functionalities and interfaces to external systems were expanded. These changes induced additional tuning actions in the application.

The IT team wrote down the lessons learned during the transition to J2EE and the deployment of the application. The intended audience is architects and developers well trained in object-orientation and Java technology.

2 Managed Environments

Application servers provide services to handle mundane housekeeping tasks such as authentication, authorization, persistence, concurrent updates. To fulfill their contract they request applications to respect a set of behavioral rules.

Object to relational bridges provide services to transparently persist graph of business objects. Various products are available, for example OJB, hibernate, EJB, etc. They track new, updated and removed objects and generates the corresponding statements to synchronize the database. The data object framework build on top of such bridges to track remote changes performed locally in a client before sending back the updated graph to the server to store it.

When using an object to relational bridge inside a managed environment care must be taken that both set of rules do not interfer.

OJB provides configuration hooks to connect to the underlying J2EE container. All major products are supported. The extension mechanism can be used to add another product. The team did not encounter major problems deploying OJB in Websphere and JBoss.

The difficulties were to understand the runtime behavior of the used container and how to configure it optimally.

The major source code migration problem was to configure the logging system. For example IBM Websphere is incompatible with the Jakarta common logging API. You must use log4j or the JDK logging packages. Care should be taken to provide a sound logging infrastructure to the servers. In particular dynamic change of logging level without restarting the servers is incredibly useful.

Significant problems happened when deploying the application on the company internal infrastructure. The transaction, authorization, and deployment components were still under development. Once these components were stable, the application behaved as expected[1].

1 During this phase two errors were found in the OJB layer. The OJB development team corrected them in less than ten days. The delay due to the instable infrastructure and associated troubles was at least two months.

3 Application Documentation

A project using the data object graph framework should publish application information.

- The logical database design should be described as a UML model. Reports from the MDA model can be generated to provide information about all attributes. Description of each attribute should be generated as Java comment. This approach allows developers to access the description with one keystroke in modern IDE.
- The physical database should be described as a ER model. Reports from the MDA model can be generated to provide information about columns, foreign keys and reference integrity rules. The indexes are documented.
- The non-functional requirements concerning the persistent objects should be provided. Trade-offs for critical functions must be documented.
- Physical design decisions should be documented. The rules when indexes are defined or not are of interest.
- The database should always be accessed through this application. All relevant pieces of information are available through the application interfaces. The database itself does not protect against data corruption that could be caused by others applications.

4 Data Objects

4.1 Retrieval

Graphs of connected data objects are retrieved from the database. Criteria are used to retrieve the subset of objects the user requests.

Bridges use their own implementation of collections. These classes provide convenience methods for example to track removal of objects. To avoid subtle dependencies to brides when transmitted classes over RMI, a deep copy is always performed on the graphs sent to client applications. A more efficient approach is to configure the bridge to use standard collections.

The data object graph framework provides deep copy operations and only uses standard collection classes. If standard collections are used deep copy operations are no more necessary.

The algorithm has the following steps.

1. Retrieve the original graph from the cache or the database through the services of the bridges. All objects part of the graph are now in the cache No changes happens in the transaction context.
2. Creates a deep copy of the original graph. This operation is only necessary if the collection instances have side effects[2].
3. Commit the transaction and returns the graph to the client application.

2 OJB can be configured to use standard collections. In this mode the deep copy is no more needed if the graph is not modified before transmission to the client. Currently the application is using the deep copy approach.

The advantages of this approach are

- The container transaction mechanisms can be used without any modifications.
- No new convenience methods need to be written or maintained. The data object framework already provides a deep copy function. If the middleware is CORBA instead of RMI, the deep copy is no more necessary because it is emulated through the transformation from Java to IDL[3].
- No dependencies to the used bridge are propagated to the client over RMI.
- The caching mechanisms of the bridge can be used without any modifications. If the caching is on a per session basis, no application specific locking mechanisms are necessary[4].

4.2 Update

Graphs of connected data objects are retrieved from the database. A deep copy of the graph is computed and sent to the client. Criteria are used to retrieve the subset of objects the user requests. The client modifies the graph of objects and sends it back to the server. The modified graph contains new objects and modified ones. Objects from the deep copy were removed.

The data object framework infers the set of modified objects, the set of inserted ones and the set of deleted ones.

The algorithm used to update the database must consider that the original graph could be still in the bridge cache and that the bridge is not able to distinguish the cached objects from the copied ones because they have the same primary keys[5].

The algorithm has the following steps.

1. Retrieve the original graph from the cache or the database through the services of the bridge. All objects part of the graph are now in the cache. The bridge must be configured so that no read locks are defined on any object of the retrieved graph. No changes happens in the transaction context.
2. Infers the set of removed objects by computing the difference between the original and the copied graphs. No changes happens in the transaction context.
3. Clears the objects of the graph form the cache of the bridge[6]. No objects of the original graph are anymore in the bridge cache. No changes happens in the transaction context. The deleted objects collected in the previous steps are still available[7].
4. Register the inserted objects, the modified ones and the deleted ones in the

3 The IDL to Java transformers always use standard collection classes.
4 All caching mechanisms are available. The optimal one is dependent on the application load profile.
5 Here we have a typical impedance mismatch problem. Databases use primary keys to identify rows, programming languages use handle. Therefore two instances with the same primary keys are two objects in the context of the programming language but only one in the context of the database or the persistence layer.
6 This step should always be performed, also in the case when the OJB cache is disabled. Framework algorithms should never be dependent on application specific configuration of underlying persistence layers – OJB -.
7 This operation could be superfluous with the future release OJB-1.1. The cache behavior in this version is enhanced to support complex interactions with J2EE containers.

transaction. Changes happens in the transaction context.

5. Flushes the objects in the transaction to create all database related artifacts such as identifiers and foreign keys. Changes happens in the transaction context.
6. Compute the list of changes to empower the client to synchronize its graphs with the version stored in the database. No changes happens in the transaction context.
7. Commit the transaction and returns the changes to the client application. The changes are the newly created identifiers, timestamps for all modified objects and all application specific attributes updated on server side[8].

The advantages of this approach are

- The container transaction mechanisms can be used without any modifications.
- The update mechanisms specific to the bridge can be used without any modifications.

4.3 Deletion

Graphs of connected objects are deleted from the database. Criteria are used to remove the subset of objects the user requests.

The remove operation of the bridge are used without any modifications.

The algorithm has the following steps.

1. Delete the graph of objects through the services of the bridge. Changes happens in the transaction context.
2. Commit the transaction.

The advantages of this approach are

- The container transaction mechanisms can be used without any modifications.
- The removal mechanisms specific to the bridge can be used without any modifications.

4.4 Consistency

4.4.1 Lightweight View Consistency

Data object graphs can have a lightweight representation. Each time a graph is updated, its lightweight representation should synchronized. This activity ensures that the topology and the attributes of the view are the same as of the graph. Two solutions exists.

- The lightweight object graph is retrieved from the server. The most actual version is returned.
- A lightweight factory is available to create the lightweight view of a data object graph. The factory never calls services from the server. Therefore this solution is always at least as efficient as the first one.

8 In our application company wide identifiers, status changes due to external systems, etc. are handled in the server.

The application is responsible to update its lightweight views. The framework does not provide hooks to perform this task automatically.

4.4.2 Persistence Layer Consistency

The persistence layer provides build-in consistency checks which diminish the risks of configuration errors. Here some best practices collected over projects.

- The foreign key columns should be declared as anonymous attributes. Anonymous attributes are not visible in the domain model. This approach is not possible with OJB due to the restrictions OJB put on the copy operations[9].
- The identifier column is visible. The framework uses this field to provide efficient retrieval and update operations. The identifier column is a numerical value. A unique index is defined on it.
- All attributes of lightweight objects should be declared read-only.
- All reference code attributes should be declared read-only.
- Declare all indexes in the OJB configuration file.

4.4.3 Database Consistency

Database engines are tuned software and provides powerful tools to insure data consistency.

- Define a unique index on the technical internal primary key. These indexes should also be specified in the OJB configuration file. Candidate primary keys should not be indexed because they were not selected as primary keys.
- Define integrity rules and needed indexes for all foreign keys. These indexes should also be specified in the OJB configuration file.
- Define integrity rules and needed indexes for all reference codes. Reference codes are read-only and their indexes never need to be recomputed.

The application currently implements the first and third rules. The second rule is only partially realized. These rules are always implemented in the database and have no influence on the functionality or configuration of the server.

4.4.4 MDA Approach

Model driven architecture and code generators codify best practices. They can guarantee that known consistency rules are always generated. The approach should follow the guidelines of OMG and support industrial standards such as Diocletian.

5 Implementation Considerations

5.1 Caches

The application framework provides caches to efficiently cache data object graphs. These caches can be used to provide additional functions on client side.

9 Consult the OJB documentation for a detailed discussion of the restrictions of anonymous primary or foreign keys.

- The server can have a cache to store retrieved persistent objects. The cache mechanism is implemented in the persistent layer OJB. The cache policy is configurable through the properties file of OJB[10].
- The client has a two level cache.
  - The first cache stores the graph retrieved from the server. A cache entry is generated for each object in the graph having a root capability. A root capability means that the graph below the node can be retrieved from the server. For example treaties and programs are roots in the treaty management application.
  - The second cache stores the copy of a retrieved graph. Again a cache entry is generated for each object in the graph having a root capability. This cache ensures that all views in the client manipulate the same graph instance.

The client caches are used to provide revert functionalities.

- *Revert changes*: All local changes performed by the user are canceled and the graph retrieved from the server is displayed again.
  The following operations are performed. The roots of the copied graph are cleared from the second cache. A new copy is created and cached.
- *Reload from server*: All local changes performed by the user are canceled and a fresh copy is retrieved from the server. If another user has changed the graph, the updates are visible.
  The following operations are performed. The roots of the copied graph are cleared from the first and second cache. The graph is retrieved from the server and added in the first cache. A copy is created and its roots are added to the second cache.

As an option caches can be disabled for special purpose client applications. If all caches are active the following scenarios can be derived.

- Retrieve a graph of data objects
  - The client requests the graph of objects for a root. The type and identity of the root are specified.
  - The framework checks if the root object is in the copied cache.
  - If the root is found it is returned else the framework checks if the root object is in the first level cache.
  - If the root is found, a copy is created and added to the second cache. The copied root is returned. Else the request is delegated to the server through the facade to the CORBA or J2EE application.
  - The request is delegated to the OJB persistence layer. If the object is in the OBJ cache it is returned else the graph is retrieved from the database, added to the OJB cache and returned.
- Store a graph of data objects
  - The client delegates the request to the server. The updated graph is a parameter of the call.

10 The cache implementation will be redesigned in OJB-1.1. When upgrading to this new version, cache policies used in the application should be challenged.

- The framework computes the set of inserted, modified and deleted objects in the server. These sets are sent to OJB for the corresponding insert, update and delete operations. Upon completion the updated attributes values – primary key identifier, timestamps and server side computed attributes such as company wide domain object identifiers – are collected and sent back to the client.
- The client side framework updates the graph with the updated attributes. All roots of the graph are removed from the first cache. A copy of the graph is made and inserted in the first cache. Now the first cache contains again the graph as received from the server upon completion of the store. Revert and reload operations can be executed with the expected results.

5.2 Client Updates

All insertion, edition and validation operations are performed in the client application without requesting server services. The network latency and bandwidth have no influence on local operations.

A client commit request sends the complete data object graph to the server. The server processes the changes and commits them to the database using the object to relational bridges. Update information are sent back to synchronize the client graph but not the updated graph itself. The updates are much smaller – only a few attributes for each modified object - than the graph of objects. This approach is a trade-off between amount of transmitted data and simplicity of the client side.

5.3 Server Updates

Business logic inserts, modifies or deletes objects on the server. The network latency and bandwidth have no influence on server side operations.

The business logic is often executed in one transaction context. The above described algorithms must be designed in a way to allow multiple executions on different or same graphs in one transaction. You could say that the algorithms must be reentrant in the context of a transaction.

The above algorithms are reentrant for all reasonable bridge libraries. But they are not the most per formant ones. The deep copy operations are no necessary if the bridge component keeps track of the changes in the graphs. Therefore server side retrieval and store operations without deep copy should be provided if efficiency must be increased.

Business logic using these server side operations should be implemented by experienced developers. They must keep track of transaction contexts and transaction boundaries.

Major gain in performance are achieved using the batch mode feature of JDBC. Enable batch mode in the connection configuration[11].

11 The actual version of the IBM DB2-JDBC driver used in the project as a documented bug in the batch mode feature. Once an updated version is available the batch mode should be activated for the application.

5.4 Queries

The portability of OJB based applications is greatly enhanced if the OJB query approach is used. Only this approach is supported in all current or future personalities. An application using the query classes can move to another personality with minimal changes.

5.5 Processing Resources

Sophisticated business logic and powerful frameworks have their toll on the processing resources. Here some rules to limit waste of these resources.

- In the context of a transaction, any object should be retrieved at most once from the persistence area.
- Reflection is a powerful approach to realize flexible frameworks. These frameworks should be carefully designed. Never should they request copying of available data or duplication of information to realize their functions. Each object creation and later garbage collection has its price.

6 Architecture

6.1 Constraints

The architecture for servers with container managed transactions must fulfill the following constraints.

- The data object framework should not be modified or have dependencies to application container services.
- The design should also work for standalone servers.
- Multiple middleware – RMI/J2EE, IIOP/CORBA, JMS – standards are supported.
- Clusters must be supported. The bridge must either support distributed caches or the cache must be cleared either at the beginning or at the end of each transaction.

As an example the per-broker object cache of OJB has a cache of objects associated with an instance of a session bean. The container decides when the bean should be discarded and the cache content cleaned. To insure the objects reflect the data stored in the database, the cache should be cleared before processing a service request.

The cache provides gains as soon as the work-flow of operations manipulates more than one time the same objects. In our application this situation arises when the information must be sent to external systems.

6.2 Solutions

The server architecture is based on service oriented architecture *SOA,* the locator and inversion of control patterns. The business object graph framework provides the following services.

1. Get a graph of objects with a root of a given type. The graph is not managed by the object relational bridge. Therefore the graph is a deep copy of a managed graph.
2. Store a graph of objects with a root of a given type. The graph to store is not

managed by the object relational bridge.

3. Delete a graph of object with a root of a given type. The graph is not managed by the object relational bridge.
4. Get a graph of object with a root of a given type. The graph content is managed by the object relational bridge.
5. Store a graph of object with a root of a given type. The graph to store is managed by the object relational bridge. Therefore the store function is implicit and does not require any code.
6. Delete a graph of object with a root of a given phenotype graph to delete is managed by the object relational bridge. Therefore the delete function is implicit and does not require any code.
7. Search functions should be implemented as a bridge managed query.

Currently the application implements searches in SQL at JDBC level. This approach was chosen for two years to insure portability of complex queries to environments no using OJB. SQL queries are used to realize the following functions.

- Search treaties, programs and businesses fulfilling a set of criteria
- Generate natural catastrophe online reports. The relevant scenarios fulfill a set of criteria

OQL queries are used for all other inquiries. All OQL queries are trivial and the only search criterion is the identifier of the desired object.

6.2.1 Data Sources

The server application has multiple data sources connecting to various physical and logical databases. The major part of interactions with the data sources are handled through the object to relational bridge library. A small set of operations are coded as JDBC statements. Special rules must be respected so that these statements do not infer with the bridge.

- A first approach is not to code any SQL statement against a JDBC statement. Always use the services of the bridge to implement application specific statements.
- The data source describing the database access should support either nested transaction or two phases commit XA protocol. Multiple connections can only be used in the same transaction if XA is enabled.

Two phases commit is only necessary due to operational constraints of the environment where the application is deployed. The *only* requirement we have is nested transactions. The reason is the following one. Certain business functions requires first to perform some modifications, commit them to the database, and when execute the next work unit. J2EE containers manage the transactional context therefore the only clean approach to implement the above business functions is to use nested transactions.

Two phases commit will be a welcome improvement in our application for the following work-flow. First we commit updates to the database, when we send the changes to an external system using JMS or a stagging table in another database, for

example accounting or central auditing. If the operation is not successful we must rollback the whole work-flow. Only XA approach enables an application to implement this function[12].

6.2.2 Identifier Creation

The store algorithm enables us to create efficiently in one pass all application specific identifiers for newly created business objects. Once the objects were flushed to the database any kind of identifiers can be generated and set in the associated objects. When the changes are committed to the database, the creation and the new identifiers are written to the persistent store in the same transaction.

The data object graph provides mechanisms to propagate the changes to the client and update its local copy of the data object graph without transmitted the objects over the wire.

The identifier generators are application specific. The algorithms can be tailored to fulfill all application specific requirements. Two strategies are quite efficient. The first one uses the persistent identity manager provided with OJB. The second uses sequences in the database to create identifiers. Some database provides identifier columns which generate on the fly unique values. This mechanism is not portable and does not always scales up[13].

7 Lessons Learned

The lessons learned were collected during mission critical application development during 2002, 2003 and 2004.

The open source community is slowly migrated to managed environments and J2EE containers. The tools are still under heavy development and not always ready for mission critical applications. But similar statements can be made for commercial grade products.

A pragmatic approach for OJB and the ODMG personality is the following one.

- Use optimistic locking to detect concurrent changes[14]. Pessimistic locking with multiple remote clients is not efficient and prohibits the use of stateless sessions.
- Avoid using removal aware collection. The data object graph framework handles which objects should be removed or not. OJB removal rules are rather a pain than a help when implementing the data object graph framework. OJB can be configured to use standard collection classes.
- Use no caching or enable it only at broker level. We suggest to use the broker level caching. The cost is similar and gains are huge as soon as work-flow processes accesses multiple times the same objects in a transaction.

12 The application does not implement XA for such cases because the external systems currently do not support it. But it should be stated that the actual solution allow system wide inconsistencies.

13 For example the configuration for such fields in clustered DB2 requires some expertise. Additionally before and after copying rows in the database during migration or restore operations requires tweaking of these columns.

14 The application has long lived transactions. Therefore pessimistic locking should be avoided. Please consult database literature for a detailed discussion of optimistic versus pessimistic locking approaches.

- Set lock association to read instead of write. This option enables OJB to create locks using a lazy approach. The performance gain is measurable.
- Use shareable data sources. Do not use user name or password. Otherwise containers such as Websphere will not share the data source as documented in an obscure IBM technical note.
- Currently the implicit locking is set to true. The exact reason why the over mode does not work is not inferred[15].
- Each time the application requests an explicit connection from a data source, it must close it to return it to the pool.

Use the OJB extension to the ODMG specification to register modified objects in a transaction.

```
((TransactionExt) tx).markDirty(dataObject);
```

First have a running system, second tune the system. Measure your system, analyze the results, decide how to improve it. Start again the loop.

Tuning a mission critical system requires access to professional performance measurement tools for the database, the application server, network traffic and end to end functions. Without such tools performance tuning is just guessing and has nothing to do with professional engineer practices.

Be aware of locking propagation in the database when performing operations associated with reference integrity rules. Such rules force the database to lock other records to insure that the rules are respected.

8 Next Steps

8.1 Persistence Broker

The actual version of the persistence broker API has powerful features. When the auto-retrieve, auto-update and auto-delete features are used it has almost the expressiveness of full grown object/relational standards. The only feature not provided is object locking instead of database locking. Analysis of the requirements of applications shows

- The locking is only relevant for applications where users often concurrently modify graph of data objects.
- Object locking can be replaced with a per broker cache and optimistic locking. It insures that no concurrent modifications on objects are possible and always detect concurrent modifications. The cache must be cleared at the end of the transaction or the objects fetched from the database in the next transaction to insure that objects always contain the latest values.
- The persistence broker is compatible with managed environment. It is the low level layer of OJB and therefore extensively tested.
- The persistence broker should be a factor two more efficient then high level

15 The restriction could be related to the ODMG personality.

standards such as ODMG or JDO.

8.2 JDO Standard

The JDO standard is the official successor of ODMG. Major players push this standard for POJO persistence.

The current JDO implementation in OJB is not production ready. Therefore the JDO approach is not a viable one with the current version.

9 Bibliography

[1]	Data Object Graphs A pragmatic Approach to MDA generated DTO graphs Marcel Baumann, 2003
[2]	OJB Documentation Apache Foundation http://db.apache.org/ojb
[3]	MDA Documentation OMG http://www.omg.org

10 Abbreviations

API	*A*pplication *P*rogramming *I*nterface
CORBA	*C*ommon *O*bject *R*equest *B*roker *A*rchitecture
DOG	*D*ata *O*bject *G*raph
DTO	*D*ata *T*ransfer *O*bject – the older terminology for this pattern was value object – The pattern is sometimes called Transfer Object EJB *E*nterprise *J*ava *B*eans
IDE	*I*ntegrated *D*evelopment *E*nvironment
J2EE	*J*ava 2 *E*nterprise *E*dition
J2SE	*J*ava 2 *S*tandard *E*dition
JDO	*J*ava *D*ata *O*bject
MDA	*M*odel *D*riven *A*rchitecture
OJB	*O*bject *J*ava *B*ridge
ODMG	*O*bject *D*ata *M*anagement *G*roup
OMG	*O*bject *M*anagement *G*roup
O/R	Object / *R*elational
POJO	*P*lain *O*ld *J*ava *O*bject
UML	*U*nified Modeling *L*anguage
USDP	*U*nified *S*oftware *D*evelopment *P*rocess
XP	E*x*treme *P*rogramming

11 FAQ

11.1 Configuration

1. Which is the best approach to create OJB configuration files?
   Use a MDA approach. Model your application with UML and use generators to create the database schema and the OJB mapping file. The web site of OJB lists a set of design tools.
   The authors recommend such an approaches. Organizations – OMG – and universities are publishing more and more about MDA. The first experiences show a gain of productivity especially for iterative development cycles and to maintain the applications. Being a mainstream technology well-trained engineers can be found on the market.
2. How should the data source be configured with OJB?
   Use the standard configuration. The data source is sharable, read committed, no two phases commit. Only if you really need two phases commit you should use it.
3. How does OJB uses the indexes declaration?
   We do not know. At least their definition does not seem to disturb OJB.

11.2 Programming

1. Which approach should we use to write complex queries?
   Use the query by criteria interfaces. This advice should only be follow if you do not plan to use other O/R tools. Otherwise stick to JDBC SQL code, which is portable but less legible and maintainable.

   ```
   PersistenceBroker broker = ((HasBroker)tx).getBroker();
   QueryByCriteria query = ...
   Collection objects = broker.getCollectionByQuery(query);
   ```

2. How should reference codes be stored in the database?
   Two approaches exists. First you can define a table for each reference code type. This approach is simple but the number of tables grows linearly with the number of types. For complex applications deployed in various environments the burden for the database administrator can become painful. Second you can define a unique table for all reference code types. OJB provides mechanisms to store instances of different classes in the same table. The management of the codes is a lot simpler. Views can be provided to help report writers working at the database level. The drawback is that the implementation of reference integrity rules is more complex and slightly slower. In any case the reference codes should be set to read-only.
3. What is the pseudo-code for the store operation of a data object graph?
   For the complete source code please consult the CVS repository.

   ```
   NarrowTransaction narrowTx =  (NarrowTransaction)getCurrentTransaction();
   // call DOG to update the inserted, modified and deleted objects
   // the flush operation writes to the database. All database generated fields are
   // available
   narrowTx.flush();
   // updates application specific identifiers and fields
   ```

```
        updateReferences();
        // collect the updates and returs them
        return storeVisitor.getUpdateInfos();
```

11.3 To Do

- Find out the difference between using mark dirty and acquire write lock to persist a modified data object.
- The DOG framework has complete knowledge which objects have been inserted, modified or deleted. Therefore the implicit locking could be disabled to speed up the performance of OJB. When retrieving graphs the implicit locking should be active to automatically retrieve all layers of the graph.
- Test if the retrieval of a set of objects is one select or 1 + n select where the first one retrieves the set and one select on the identifier is performed to find out if the object exists.
- Test if spurious select statements to verify that an already read object still exist in the database are generated.
- Evaluate if refactoring the facades in the DOG framework is worth the costs. In other words do we have ROI arguments.

12 Annexes

12.1 Weaknesses of OJB

The apache open source project OJB has strengths in its design and the variety of databases and environments it supports. The project has also weaknesses which should be documented to avoid disappointments.

- Releases are slow to be published. Open source projects have seldom a predictable release stream with deadlines.
- Commercial support is currently not available as maintenance contracts. It is possible to contract some developers of OJB but no offering from a company exists.
- Experience shows that each time a new version is released, the projects we are working on experience one to three errors[16]. Once these errors are isolated the response time to correct them is quite good – between a few days up to two weeks -.
- Testimonials show that complex applications were developed and deployed with success – see http://db.apache.org/ojb/references.html -. But none of the reports describe the configuration we are using in some of our projects.

12.2 Alternatives to OJB

Various object relational mapping tools and persistence layers are available on the market. Here some information about alternative products. The classification defines a set of standard approaches – J2EE EJB, JDO, ODMG – and non standardized approaches. The products for each approach is either commercial or open-source.

- *J2EE EJB:* The enterprise Java beans are the official persistence mechanism for J2EE applications.
  - *EJB 1.x:* The standard is obsolete and replaced with the newer and more powerful version 2. All references insist that new projects uses the newer standard. The older one should only be considered for legacy purposes.
  - *EJB 2.x*: This standard defines the official persistence mechanism for J2EE based application. Work is currently under way to defined the revision 3. This revision will integrate the plain old Java object POJO approach.
- JDO
  - KODO – commercial offering -
  - Open source JDO implementations are slowly available. But reports describing their use in mission critical applications are not available.
- ODMG

16 In one iteration the sequence generator for DB/2 database was erroneous for the version we were using. In the next iteration the object in the envelope of a transaction was not replaced with a newer one – with the same identifier – in a container manager persistence configuration. In the same iteration we found out that calling mark dirty instead of lock for write improved the performance by a huge factor.

  - POET – commercial offering -
- Non standard solutions
  - TOPLINK – commercial offering – The company who developed the product was bought by Oracle.
  - Hibernate – open source – is a well-known persistence layer. It is the preferred library for the J2EE JBoss product. The API is custom. A partial support of ODMG is provided. Future development is heading to EJB3 to support this standard in JBoss.

An entry point for extensive information about mapping products can be found under - http://c2.com/cgi-bin/wiki?ObjectRelationalToolComaprison -.

12.3 Cache Implementation

Readers requested a description of the cache implementation in the initial version of the DOG framework.

- *Facade Manager* is responsible to manage all facade instances. The manager provides the facade responsible for a specific data object type.
- *Facade* The facade interface defines an abstraction to access data objects of a specific type and to describes the type. The type information is used to access all roots of a data object graph. The services of the class can be grouped in three sets.
  - Services to describe the capabilities of the root data object type managed through the facade instance – data object class, lightweight class, etc. -.
  - Services to read, update and remove a graph of data object. The root of the graph must be of the type managed in the facade instance.
  - Services to manage the caches defined in the facade instance.
- *Facade Implementation* provides a default implementation of set of the methods of the facade interface. The class is abstract.
- *Facade without cache* provides a default implementation of a facade without cache. The class was deleted in a next version of the framework. The visitor pattern is used instead.
- *Facade with cache* provides a default implementation of a facade with a cache. The class is abstract.
- *Client facade implementation* provides an implementation of facade with cache for the client. The class inherits from facade with cache and implements the abstract methods retrieving the children of data object instance. Children are only returned if they are root objects – only treaty and period have children -.
  The retrieve methods are hard coded and do not use model information to retrieve the root children of a data object instance.

The above classes are used for two years in the application without any troubles. If a refactoring is once performed, the classes could be replaced with a global cache for the client. The graphs could be traversed using iterators. The type information is still needed to decide if a data object type is a root or not.

12.4 Meeting with Armin Waibel

A telephone conference was hold Tuesday, 07 September 2004.

- How should we handle J2EE container initiated rollback?
    - Set rollback only on container does not throw a container object. The session bean should throw an exception – for example remote exception – to inform the client application. The internal data structures should be cleaned if a rollback is executed.
    - Avoid batch mode in the JDBC driver. Possible errors are still in the actual release of OJB-1.0. *Our application do not use batch mode because we must use an IBM JDBC type 2 for DB2. This driver version has bugs in the handling of the batch mode.*
- Sequence manager for native identity had an error in situation where multiple session beans because each broker has a separate instance of the sequence manager.
    - Mister Armin Waibel will release a patch in CVS.
- Should we close the connection when requesting it explicitly from the data source?
    - Yes. Data sources in managed environments does have the problem that no commit can be programmed. OJB has a wrapper to block all prohibited operations. But sometimes OJB cannot close the connection because it is not informed from the container about completion – for example with JBoss -. Similar symptoms are possible in other containers.
- Our application has trouble reading a tree of data objects, extracting some data, throwing them away, and writing a clone of the same tree.
    - The problem is currently unknown. Armin will try to reproduce it.
- Queries always generate an order by on the primary key if no order clause was defined. These operations are expensive in DB2.
    - This situation is probably an error in the query management in OJB. Mister Armin Waibel will look at it with the responsible developer. The sort is probably defined to support paging and limitation of results when executing a query.
- Cache management definition for managed environments.
    - Cache per broker or empty cache should be used currently. Shared caches must be distributed but still allow dirty reads.
- Auto-retrieve, auto-update and auto-delete
    - The flags should be set to true, false, and false. The flags must be set manually in the repository file.
    - ODMG and persistence broker performs a propagation of inserts if a new tree is added to the database.
- ODMG versus persistence broker API
    - The application could be run using the broker API. The performance increase is about 50 %. The auto flags – retrieve, update, delete - are

available to emulate a behavior very similar to the one of the ODMG API.

    - The persistence broker also support insert, delete and update propagation through a tree of data objects.

- Are new releases planned in the future?
    - The minor release 1.0.1 was released the 8th September 2004. The minor release 1.0.1 should be released end October 2004.
    - The release 1.1 is the next major release. The release date should be end of 2004.
    - ODMG interface is less used and will be replaced in the future with JDO.

We would like to thank Armin for the information and the support.

12.5 Technical University Discussions

The meeting was held Wednesday 2nd February 2005 in Zug.

- Why optimistic locking is used for detecting concurrent changes in DOG?
    - The design assumes that clients perform complex activities. To avoid long living transactions the optimistic locking approach is used. If your application requests over approaches you must modify the templates.
- How are transactional contexts managed in DOG?
    - The transactional context is managed through the framework and pluggable transaction managers. Once version used for CORBA server manages the transactional context for each CRUD operation performed on a data object graph. A second version used for J2EE server delegates transaction management to the J2EE container.
- What is the purpose of the lock manager?
    - The lock manager is an application level object lock manager implementation. The algorithm knows how to retrieve all root nodes of a graph. The lock manager is optional and should only be used when performance is too low in a distributed J2EE environment.
- Are shallow objects supported in the actual version of DOG?
    - Yes, but the developer must program the details. See the *DOG* design document for a discussion which operations are legal on shallow objects.
- Which are the work package to support .NET?
    - Adapt the Velocity templates to support .NET business object written in C#.
    - Write the equivalent of the DOG framework on the server side.
    - Plug the DOG framework to OJB.NET.
    - Write the equivalent of the DOG framework on the client side. The client will talk with server using .NET remoting features.
- Integrity rules templates should be corrected to generate usable code. The corrected templates should be delivered.

    - Trunk 0.3.x contains integrity rules template working with MySQL – CVS check-in 10th February 2005 -.
- N to M relations are not working correctly.
- Import declarations for indexed properties are not generated in certain situations.
    - Could not be reproduced with the actual test cases. We are looking to define new ones to identify the problems.
- Constructors with parameters are not correctly generated.
    - Due to the limitation of this option allowing only the definition of one additional constructor the associated tagged value was removed. Work is under progress to define a more general solution.
- Generation of changeable and vetoable events do not follow the constraints of the associated tagged values.
    - Trunk 0.3.x contains updated template code. Refactoring of tagged value names was not applied to some templates therefore regression problems occurred.
- The accept method should call the accept method of simple properties having a business object as type.
    - Trunk 0.3.x contains updated template code – CVS check-in 10th February 2005 -.
- The abstract visitor functor has an erased type for the functor instance. Is it really necessary?
- Could the different frameworks – MDA generator, explorer framework, data object graph framework – stored in different directories?
    - The MDA generator already has a seperate source directory. The DOG and explorer frameworks are stored in the same source directory but are distinguished through their package names. The application framework is ch.bbv.application.*, the DOG framework is ch.bbv.dog.*, the explorer framework is ch.bbv.explorer.*. Currently no plan exist to further seperate the frameworks.
- Could more complete UML diagrams be provided as overviews?
    - Yes, an updated model with more class diagrams and information will be provided.