Market Requirements Document

Feature Name: Graph-Optimized Associations

Version 2: Date Submitted: 04/13/10 Completed By: Leon Guzenda
Version 1: Date Submitted: 04/12/10 Completed By: Leon Guzenda

Description of the Problem

Objectivity/DB associations provide an efficient mechanism for maintaining multi-cardinality links between objects. However, they were designed for applications that almost always look at the data in objects that are visited when a navigation iterator visits them. We are now increasingly encountering classes of problems where rapid traversal of the links between specific objects in a graph is more important than looking at the data in the intervening objects. For instance, determining whether or not there is a path between two objects, regardless of the association (and hence, object) types involved.

Description of the Requested Feature

An optional Graph-Optimized attribute of standard (dynamic, inline short and inline long) associations. A Graph-Optimized association will be implemented to make traversal of a graph, particularly for the type of operation described above, as efficient as possible. There may have to be trade-offs between existing, data-optimized and the new graph-optimized mappings.

It would also be convenient to be able to group edge attributes with a particular link, e.g. to specify the Number_used and Size of the wheels shipped with a Vehicle in a Vehicle-to-Parts association. This can be done with objects and associations, but it is somewhat clumsy. The edge attributes should be physically located with the association VArray element.

Part of an existing feature or does it require another feature, if so, which one?

- This would be an enhancement of an existing feature.

How is this problem being solved now, and why isn't that acceptable?

Some developers have resorted to using ooVArrays of OIDs in order to avoid any perceived overheads of the standard association mechanism. This should not be necessary, as associations are actually mapped to underlying Varrays. However, association VArrays cannot be pre-sized and performance degrades as they become large slots. Both of these problems may be fixed as a part of the Segmented VArray project.

Also, as described above, handling edge data as a part of a traversal or query is currently clumsy.

What languages must support this capability?

- C++ (Essential)
- Java and .Net for C# (Soon)
- SQL++ (Eventually)

Which platforms must be supported?

- All.

Do any competitors already have this feature?

- None of the ODBMSs is as efficient at handling cross-database containers as Objectivity/DB. However, there is a growing market for graph databases, such as Neo4J and Hypergraph. They are aimed at providing fast graph traversal as a front end to RDBMSs. Neo4J is particularly efficient and fast, so we should benchmark against it.

Customers who require this feature

- The Intelligence, security and defence markets.
- It would speed up many aspects of the NGC (TRW) application.
- Social networking and relationship analysis applications.

Revenue at risk, or which could be won

- It is sobering to realize that sites such as Facebook, linkedIn and Plaxo could easily have been developed with Objectivity/DB as the underlying storage engine.

When is this required?

- Release 10.2, or at some point prior to R11.

Additional Notes

1. Related Material

We will also need:

- New Quality Assurance material.
- Updated documentation, training and web based training.

2. Implementation Recommendations

Associations are held in VArrays (Storage Manager slots) that generally reside in the same physical page as data objects. The Link Hunter demo has shown that densely packing the link information can significantly speed up traversal of graph structures when it is not necessary to look at the data within each traversed object.

If the developer can specify that the association Varrays are not mixed with object data then the link data will be stored optimally. However, traversing a link currently involves fetching and opening the target object before opening the association VArray for onward traversal. So, it would be convenient to store the OIDs of the target association Varrays rather than the target objects.

If only this change was made it would be possible to reach a VArray and traverse onwards, but it would currently be hard to find the owning object. So, the new mapping should store the OID of the owning object in addition to the link data. One possible implementation is described in Appendix A. It should be possible to leverage work on segmented or nested VArrays in order to meet this new requirement.

APPENDIX A – A Possible Implementation.

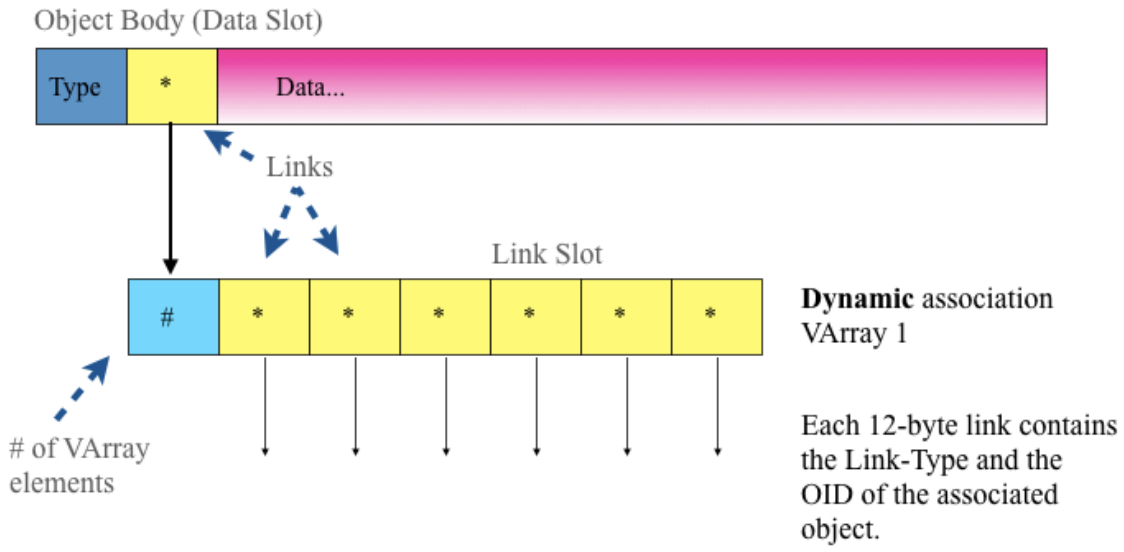
The following images have been captured from a presentation on this topic.

Current Object and Association Layout

- Provides direct access to the object body from C++ without the need to move it out of the cache page.
- Supports multiple VArrays per object.
- Uses a dedicated slot for the dynamic association VArray:
  - Each element is a {Link-Type, long-OID} pair.
- Uses floating slots for each in-line association VArray:
  - In-lines can be 4 or 8-byte OIDs.
- VArrays may be in separate pages.

Current Object Layout (1)

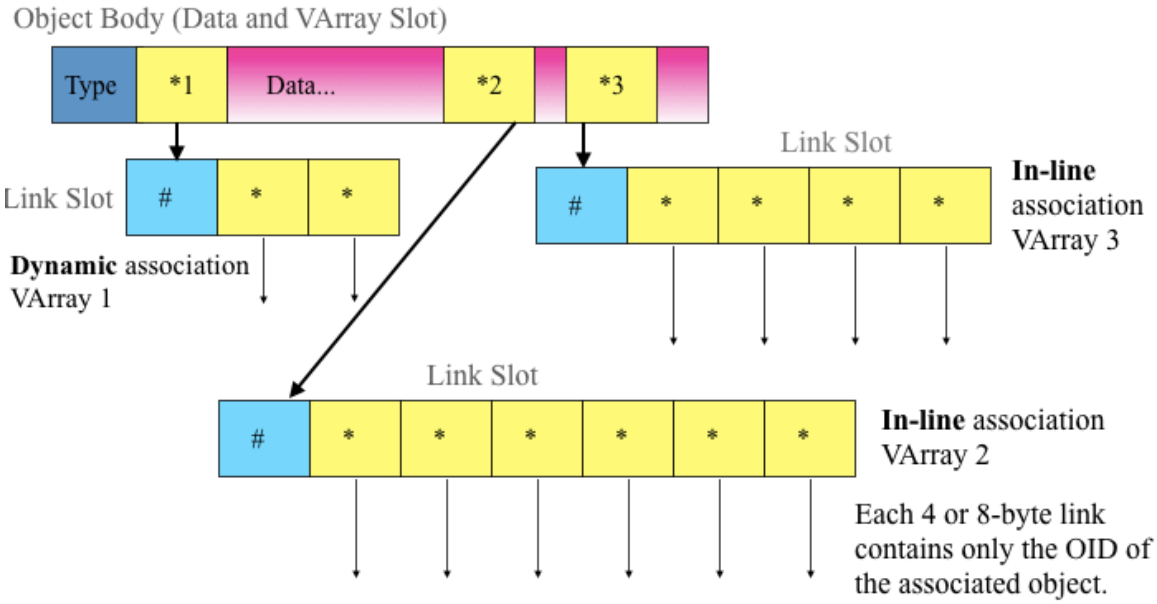
A typical C++ object with dynamic associations



PROBLEM: The data reduces the amount of link data that can fit into a page.

Current Object Layout (2)

A typical C++ object with dynamic associations and two in-line association types.



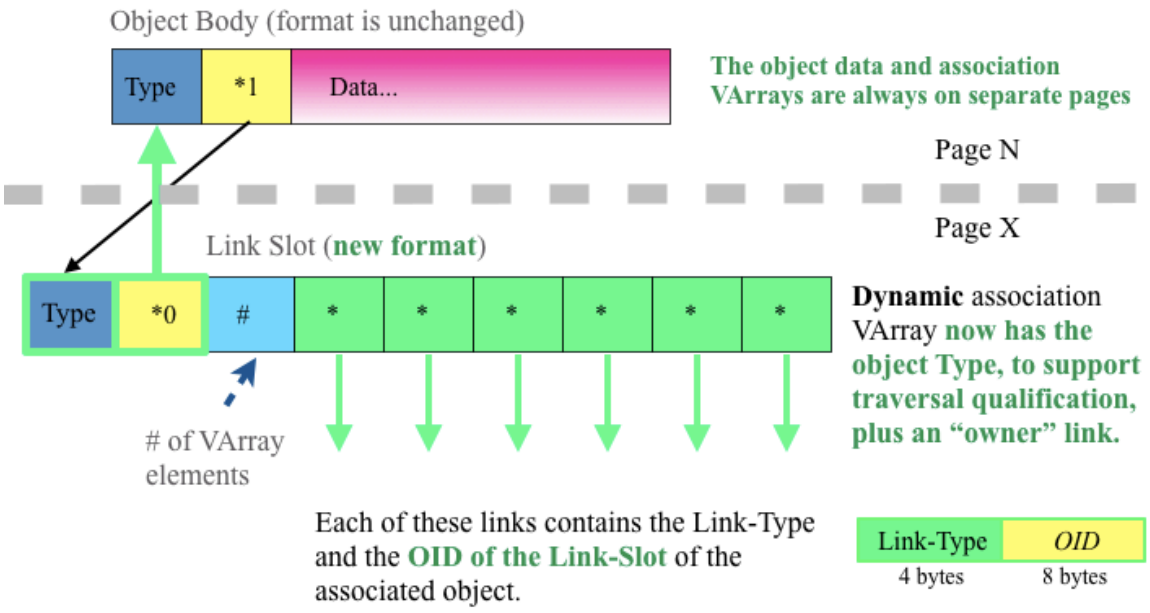
PROBLEM: The data reduces the amount of link data that can fit into a page.

Graph Databases

- Graph traversal requires very dense packing of node identification and links (edges).
- An in-line association VArray is very densely packed However...
- **The object body can be large.** It's at least 8 bytes plus data. It can force the links out of the page, increasing the number of I/Os required.
- On the other hand, if we build another navigational structure that just stores an OID and the links, that's an extra 8 bytes anyway, so, why not use those 8 bytes in a more productive way...

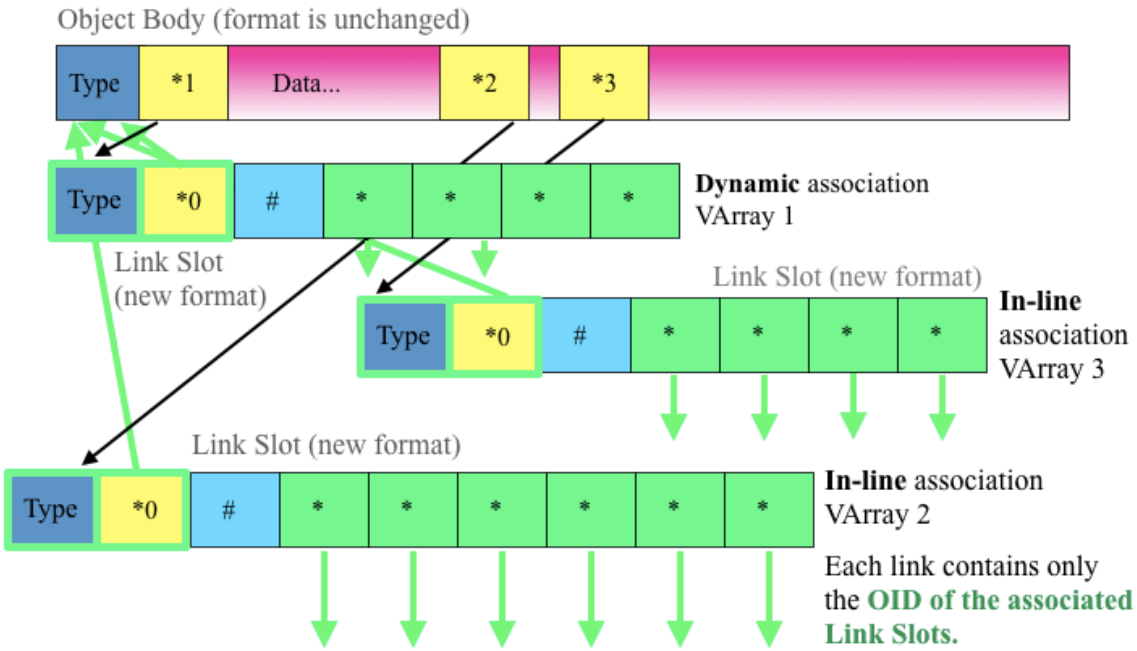
Proposed Object Layout (1)

A typical C++ object with dynamic associations



Proposed Object Layout (2)

A typical C++ object with dynamic associations and two in-line association types.



Pros & Cons

PROs

- Much more link data can fit into a page:
  - **The association links are to other association VArrays, not the object data.**
  - The slot housekeeping is used to distinguish between old and new association slot types.
  - Both can co-exist, determined in the DDL or at runtime.
- Navigating between objects, or finding the shortest path between two objects, will only read in the link pages.
- **There will be no need to build supplemental structures for rapid graph traversal.**
- The scan() operator will be faster because there will be more object bodies in each page.
- Having the "owner" link from the VArray slot to the data slot will improve resilience.
- Only requires relatively minor, localized, kernel (Object Manager) modification.

CONs

- We've added 8 bytes to the storage overhead (for the Type and the link to the data slot):
  - We could make the Type field optional, saving four bytes.
- Access to object data once you've navigated to the link VArray will involve an extra I/O:
  - However, with the current scheme there's also a significant probability that, after navigating to an object, the association VArray is found to be in a different page, requiring an extra read operation before onward navigation can proceed.

Summary

- Most graph traversal applications would use in-line associations, which can be stored in the new format, so the links will be very densely packed.
- Traversal of association links will be extremely fast.
- New format Association VArrays use the OIDs of other association VArray slots, not data slots.
- They also contain an "owner" link and an optional object Type field.
- We could allow the user to store a link weight, or other link data, in the link VArray or an additional VArray.
- We can distinguish between "old" and "new" object layouts in the slot housekeeping. They can co-exist in the same container.
- The user can define the format to be used in the DDL or at runtime.
  - There could also be a tool for converting between the formats.