MARKET REQUIREMENTS DOCUMENT

Feature Name: Objectivity Scalable Collections

Version: 3 **Date:** 1/31/2005
Version: 2 **Date:** 1/26/2005
Version: 1 Date: 5/11/2004

Completed By: Brian Clark and Leon Guzenda

Description of the Problem

Background

Objectivity has implemented various collection classes in a piecemeal way. Keyed objects (hashed) were part of the original product but these worked well only for small numbers (100-1,000) of objects and when the hash table could be pre-sized. When the hash table had been resized performance would degrade (lookup overflows).

Maps were added later (ooMap) to overcome some of these problems. The hash table is dynamically extensible, once resized performance is reliable and predictable. Maps are OK for larger number of objects (1,000 – 100,000).

Persistent STL support was added in the C++ binding which increased the number of types of collection classes available.

With the Java binding we introduced the interoperable scalable collections, which implemented the Java collection class interface.

The Problem

Performance: Various problems have been reported with the performance of the scalable collections. Performance is not consistent or predictable as the collection grows. Tuning for performance is not obvious.

Scalability: The collection classes have not proved scalable either for small numbers (too much overhead) or for large numbers (performance unpredictable and too slow).

Clustering: There is little or no control of placement of the collection objects, containers and databases, particularly in a replicated environment.

Description of the Requested Feature

Functionality

We need a set of collection classes that provide the functionality of the existing classes, implementing the Java collection class interface, that are interoperable across supported language bindings (Java and C++), that are scalable both upward (large numbers) and downwards (small numbers), and that perform in a reliable and predictable manner.

To do this we need to meet the following requirements:

- Insertion/lookup time should be near constant whether it is the first, last or middle of the collection.
- Insertion time of the first n should be close to the insertion time of the next n, etc.
- Provide use/test cases demonstrating the above at say 1,000, and 1,000,000, and 1,000,000,000 objects for each collection class.

We need the following types of collection classes:

- Treeset
- Hashmap
- [the existing list of scalable collection classes]

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

Tunes an existing feature.

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

The current scalable collections are not scalable to the quantities we are already experiencing at our VLDB customers.

What languages must this capability support?

Java and C++.

Which platforms must be supported?

All.

Do any competitors already have this feature?

Yes (equivalent)

Customers/Prospects who require this feature

Most VLDB customers are expected to need collections of one sort or another.

Revenue at risk, or which could be won

VLDB customers

When is this required?

Release 10

Additional Notes

1. Should look at existing scalable collections re-use.
2. Should look at ooVector.
3. Should look at “segmented VArray”.
4. We’ll need additional Quality Assurance material.