Market Requirements Document

Feature Name: Relationship Analytics

Version 4: Date Submitted: 01/11/10 – name chaged to Relationship Analytics

Version: 3 Date Submitted: 08/05/09 – Name changed from Generic Iterator to Connection Explorer

Version: 2 Date Submitted: 10/12/07
Completed By: Leon Guzenda, with input from Technical Services and Engineering.

Version: 1 Date Submitted: 09/08/05 Completed By: Leon Guzenda

Description of the Problem

Many intelligence and telecom networking applications would benefit from the ability to rapidly traverse a network of objects without needing to know the kinds of links that exist beforehand.

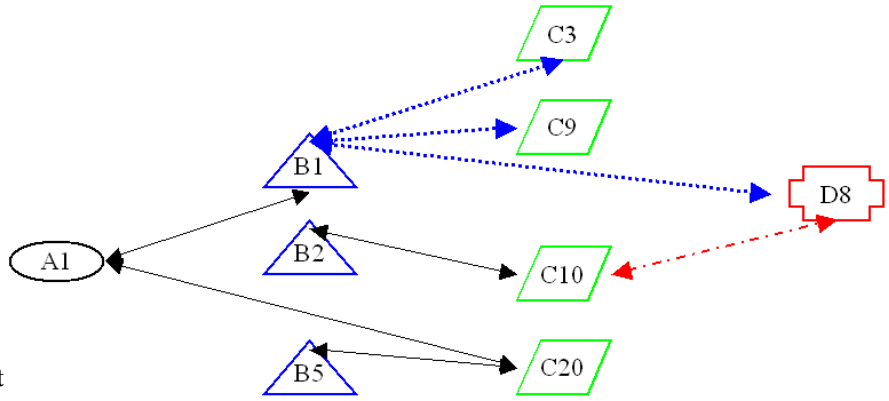
Users currently specify the type of the association to be iterated over. The kernel can perform a transitive closure iteration in support of atomic propagated delete and lock operations, but it is not possible for an application to do transitive closure without knowing or determining the kinds of links between objects.

Description of the Requested Feature

A new link iterator, inheriting from ooObj, would perform a depth-first transitive closure from an object to all objects linked to it, directly or indirectly, regardless of association type or association format (dynamic or inline). Embedded Refs (in the object body or in its VArrays) should be regarded as links, but the API should allow the caller the option of iterating using only associations.

Example: The diagram below shows a network of connected objects.

Suppose that iteration starts from A1. The iterator would return (not necessarily in this exact order) B1, C3, C9, D8, C10, B2, C20 and B5.



There should also be a link checking API which returns true or false depending on whether or not there are any links between two objects. In the above example it would return true for a check for a link between any two objects.

Users need to be able to reject certain paths as shortest. For example, given a source and destination object, there should be an API to iterate over representations of multiple paths between them in distance order, starting with the shortest.

The iterator and the link check should not go into a loop if there is a ring of connections between objects, e.g. A to B to C to A.

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

- Enhances iteration features.
- Should be incorporated into the Parallel Query Engine and Advanced Query features.

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

It can be solved by application programmers, but only with the use of Active Schema and by developing the transitive closure algorithm.

What languages must support this capability?

- C++
- Java
- SQL++

Which platforms must be supported?

- All platforms.

Do any competitors already have this feature?

- No, but some have transitive closure for a single association /relationship type.

Customers who require this feature

- Relationship hunters (intelligence community and telecom network applications).

Revenue at risk, or which could be won

- We frequently talk about the ability for Objectivity/DB to “find needles in multiple haystacks”, but we don’t actually supply any powerful navigational search capabilities. This capability, used in conjunction with the Parallel Query Engine, could be a very powerful selling feature.

When is this required?

- As soon after Release 10 as possible.

Additional Notes

1. Related Material

We will also need:

- Marketing collateral, including a Press Release.
- Technical Publications.
- Release Note.
- Extra Quality Assurance Material.

2. Uni-directional Links

Finding links back to an object from objects that have a uni-directional link to it can be deferred until later. The class instance feature of the Parallel Query Engine will make it faster, but it could still take a long time.

3. Typed Transitive Closure Iterator

We should also consider providing an iterator that performs a transitive closure (loop checked) navigation via a named association type.

4. Link Analysis

The link checker could be supplemented with an API that would return a list of the OIDs of the objects involved in links between any two objects.

5. Ad Hoc Query Support

Declaritive support for ad hoc queries, probably implemented as a part of the Enhanced Object Qualification project, can be delivered after Release 10.

6. Scalability

The loop checking algorithm should work for extremely large federations, i.e. it must not be able to run out of virtual memory. One way to implement this would be to use a container that is deleted after it has been used.

7. Types of links

The first implementation is limited to associations because their purpose is clearly to connect objects. Concerning other types of link:

- VArrays of (or including) references could be optionally included in the iteration or connection check. They will not be searched by default.
- Links within embedded objects can be treated like VArray references.
- Collections can be left until later as they often have semantics such as ("I found these when I ran a query concerning issue blah blah blah") rather than a direct connection. There is already an API that can determine whether an object is in a collection and to iterate over a collection.
- There is no need to support lookup keys.
- There is no need to use template specialization to capture the type filter of result objects, i.e. it will be sufficient to use ooItr(ooObj)::set_typeN() or an analog.
- The user needs the ability to analyze the object of iteration and decide if navigation from the object is needed, ie., at the time of visitation of a qualifying object, the application may make a dynamic determination to direct the iterator not to further traverse links from this object. It would seem that this does not require any new mechanism in the iterator API as users can do this with current iterators. However, declaritive queries could use this feature.

8. Candidate Requirements for Later Releases

- Visitation of an object should entail a way to obtain a representation of the links traversed to reach it.
- We need the ability for users to specify the target types, which can be used to improve efficiency by avoiding paths that cannot lead to that type.
- Allow users to be able to specify how a link's cost is determined. As opposed to considering it as just having a weight of 1:

    - For example, an application could assign a weight to link specifications such that two hops with one kind of link are equal to one hop with another kind for purposes of computing shortest path.
- Make it possible to limit the search depth.
- Make it possible to specify whether a depth-first or breadth-first search algorithm is employed, defaulting to breadth-first .
    - Breadth-first search causes nearest objects to be visited first, but will probably have higher virtual memory cost where the intention is to visit large numbers of distantly-related objects.
    - Depth-first search is more useful for algorithms generating a visualization of the paths.
- Multiple visitations of the same object using different paths should be suppressed by default but be enablable. Note that the user needs to be warned that iterating onward from that point will cause the algorithm to loop forever.