Connection Explorer Capsule Specification

Purpose

This Capsule provides a starting point for applications that need to find the shortest or all routes between two or more objects, or to navigate outward from an object regardless of association type.

Functionality

The Connection Explorer capsule will ultimately be able to exploit underlying kernel functionality that uses standard associations and exploits schema information to predetermine potential paths between objects of a given class. The capsule provides tools or methods for:

- Iterating from an object instance to all associated object instances, regardless of type and association link type, across multiple "hops". In a simple case this would be equivalent to exploding or imploding along a many-to-many link on a single object class. Notes:
  - The iterator must detect loops or be directed to turn off loop checking if some other mechanism has made it impossible to create them.
  - It should be possible to specify a maximum number of hops, which is useful in support of applications that produce graphical depictions of relationships.
  - It should be possible to specify depth first or breadth first iteration.
- Generating a Potential Path Map (PPM) for all possible pairs of object classes in a federation. The PPM indicates the shortest possible route and all possible routes as a list of association link types. See Appendix A for an example of a PPM. The algorithm used to generate the PPM is described in a separate document in order to protect Objectivity Intellectual Property Rights.
- Iterating over the shortest path between two or more object instances, using the PPM to search the most likely types of path first. The iterator may start from any of the endpoints, but may be constrained to present results as if it had started navigating from the first object specified. See Appendix B for an example.
- Simply returning a True/False indicator in response to a request to see if there is a path between two or more objects. See Appendix B for an example.
- Iterating depth first along all paths between two or more objects. It uses the PPM to avoid inevitable dead ends. See Appendix B for an example.
- Iterating breadth first along all paths between two or more objects. It uses the PPM to avoid inevitable dead ends. See Appendix B for an example.

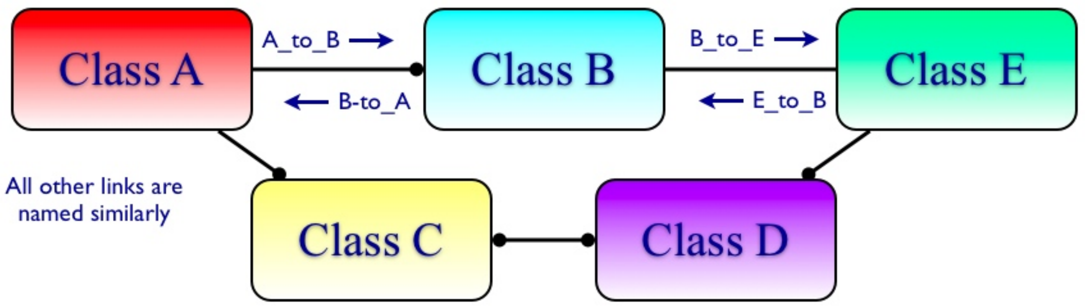
Platforms and Languages

- Windows and Linux.
- C++, Java and C# (later).

Suggested Pricing

- $500 for a single develop license plus standard Objectivity/DB developer and end user licenses.

Appendix A - An Example of a Potential Path Map (PPM)

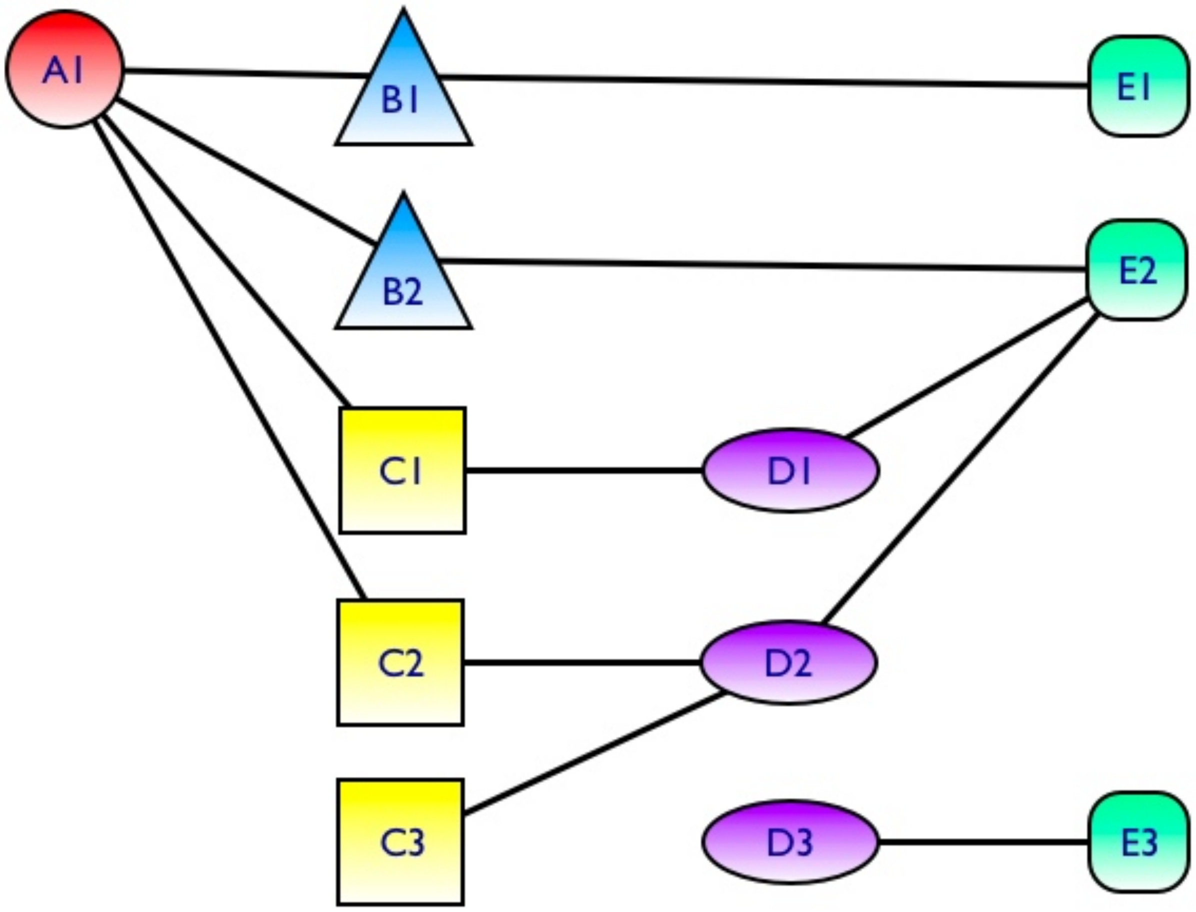


Only two pairs are shown for simplicity. There would be 20 in all.
Note how the cardinality of the links determines the fastest search path.

Pair	Shortest	Others
A, B	B_to_A	1. A_to_B 2. A_to_C, C_to_D, D_to_E, then E_to_B 3. B_to_E, E_to_D, D_to_C then C_to_A
A, E	E_to_B then B_to_A	1. A_to_B then B_to_E 2. A_to_C, C_to_D then D_to_E. 3. E_to_D, D_to_C then C_to_A

Appendix B - Iterator Examples

The iterators and connection checks all use the Potential Path Map for guidance.



- Shortest route from A1 to E2: E2 to B2 to A1 (optionally) present in A1 to E2 order
- Are B1 and D1 connected? Yes.
- Are C3 and D3 connected? No.
- Are A1, D1 and C3 connected? Yes.
- Shortest route from E1 to D1: E1 to B1 to A1 to C1 to D1.
- All routes from A1 to E2 depth first: A1 to B2 to E2; A1 to C1 to D1 to E2; A1 to C2 to D2 to E2.
- All routes from A1 to D2 breadth first: A1 to C1 to D1 to E2 to D2; A1 to C2 to D2.
- Depth first iteration from A1: A1 to B1 to E1; A1 to B2 to E2 to D1 to C1, then E2 to D2 to C2, then D2 to C3; A1 to C1 to D1 to E2 etc.
- Breadth first iteration from A1: A1 to B1; A1 to B2; A1 to C1; A1 to C2; B1 to E1; B2 to E2; C1 to D1; C2 to D2; D1 to E2; D2 to E2.