Next Steps for the COIN-OR Open Solver Interface

Lou Hafer[1] Matthew Saltzman[2]

[1]Department of Computer Science
Simon Fraser University

[2]Department of Mathematical Sciences
Clemson University.

INFORMS Annual Meeting
Austin, Texas
November 10, 2010

Outline

- What's wrong with the current API?
- Dynamic Plugin Architecture
- Factories and Run-Time Instantiation
- Callbacks: Function-Pointer and Object-Based Approaches
- Extending the Basic Interface: OsiSimplex, etc.

What's Wrong with the Current OSI?

The current OSI architecture is the original one from 2000. It has proved to be useful in many contexts, but it has several properties that make it difficult to use and maintain.

- Back-end interfaces are created by inheritance from a base class (front end).
  - Front-end and back-end interfaces can get out of sync.
  - Too many tasks are implemented in the back end.
  - Extending interfaces is difficult.
  - Separation between instances and algorithms is problematic.
- Solver interfaces are required at compile time.
  - Control of which interfaces are needed is handled in source code, or by preprocessor directives (`#ifdef`).
- Solver libraries are required at link time.

Plugin Architecture for Dynamic Solver Engine Linking

- Decide which back-end interface and solver engine at runtime.
- Load just functions needed.
- No need to include engine library references with binary distributions.
- Windows DLLs and *nix static, shared, and dynamic libraries can be handled.
- C++ and C interfaces are possible. With C++, all class information is exposed across the interface. With C, objects can be wrapped to pass across.

Callbacks and Event Handlers

Opprtunity for the user to collect information about algorithm progress or take action at certain points in the algorithm or on the occurence of certain external events.

- C solvers allow the user to register functions with the solver to be called at those points.
- C++ solvers call virtual functions at those points. Users can derive replacement functions to achieve the desired result.
- A portable callback framework is a challenge. Full callback flexibility probably requires exposing the solver engine's callback interface.
- Portability for callbacks using the intersection of common callback interfaces can be done.
- Hiding the C mechanism from C++ programs is possible. Hiding the C++ mechanism from C programs is possible if the set of actions is fixed (e.g., just abort).