The Common Component Architecture: Introduction and Concepts

CCTTSS Tutorial Working Group
http://www.cca-forum.org/tutorial/

This work has been sponsored by the Mathematics, Information and Computational Sciences (MICS) Program of the U.S. Dept. of Energy, Office of Science.

Contributors

- Ben Allan, SNL
- Rob Armstrong, SNL
- David Bernholdt, ORNL
- Lori Freitag, ANL
- *Jim Kohl, ORNL (Presenter)*
- Lois Curfman McInnes, ANL
- Boyana Norris, ANL
- Craig Rasmussen, LANL
- Jaideep Ray, SNL

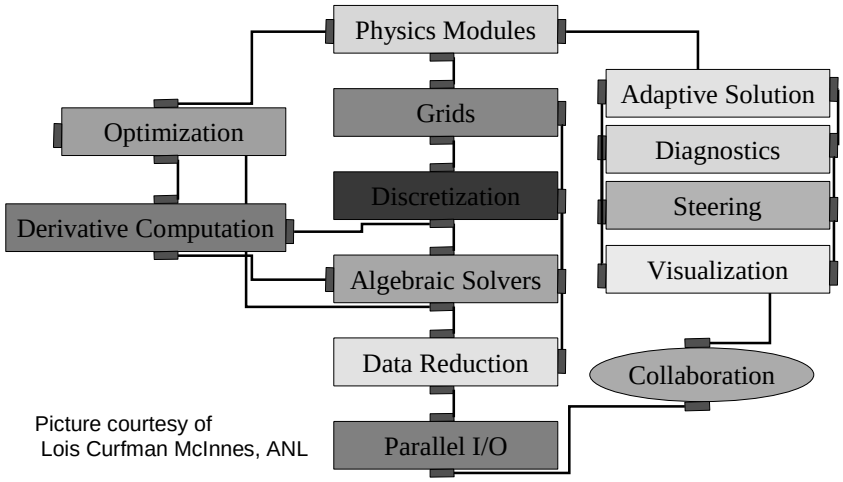
Challenges in Modern Scientific Software Development

- Desire for rapid development cycle
- Diversity of languages and tools
  - What can be used together?
  - Ability to reuse code from inside or outside organization
- Parallel computing significantly increases algorithmic complexity
  - Where do you find the expertise in all disciplines your code might require?
  - Architectural complexity and diversity of modern MPPs
- Increasing capability of computers leads to new expectations of simulation
  - Higher fidelity, greater sophistication
  - Extension to multi-scale and multi-physics simulations
  - Coupling of simulations

Component-Based Programming

- Based on OO ideas, but at a coarser level
- Components encapsulate well-defined units of reusable functionality (in OO sense, often a collection of objects)
- They interact through well-defined interfaces
  - Separates interface from its implementation: “Good fences make good neighbors”
- Components improve modularity and reuse
  - Provides for “plug and play” HPC code.
  - Facilitates exchange of components between groups
  - Facilitates interdisciplinary collaboration in a single app.
  - Allows you to focus on your area of interest/expertise
- Intended to make it easier to *compose* software into a working application
- *Not* a magic bullet
  - Does not alter algorithms: does not speed up their development, or eliminate bugs. Does not make *programming* easier

21st Century Application



Picture courtesy of
Lois Curfman McInnes, ANL

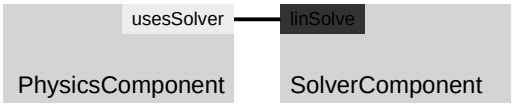
Commodity Component Models

- Component models are common in visualization environments
  - AVS, Data Explorer
  - dataflow-based
- Component-based software development has taken the commercial/industrial world by storm.
  - CORBA, COM/DCOM, Enterprise JavaBeans
- Unfortunately, these systems are not generally suitable for high-performance scientific computing
  - Focus exclusively on local & distributed computing
  - No concept of parallelism
  - High overheads, even for local operation
  - Often platform or language-specific

The Common Component Architecture

- A component model specifically designed for high-performance computing
- Supports both parallel and distributed applications
- Designed to be implementable without sacrificing performance
- Minimalist approach makes it easier to componentize existing software
- Components are peers
  - No particular component assumes it is "in charge" of the others.
  - Allows the application developer to decide what is important.

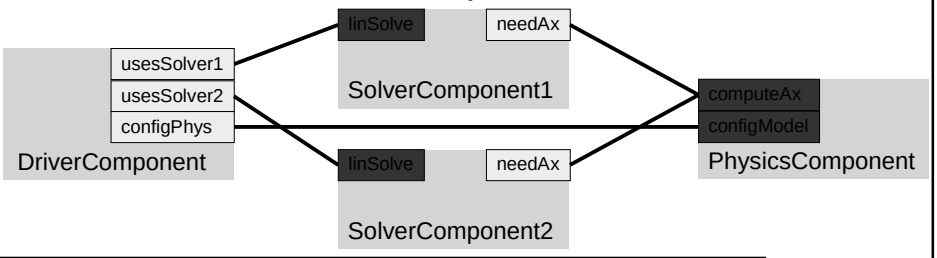
CCA Concepts: Ports



- Components interact through well-defined interfaces, or *ports*
  - In OO language, a port is a class
  - In Fortran languages, a port is a bunch of subroutines
- A given component may *provide* a port – implement the class or subroutines
- Another component may *use* that port – call methods or subroutines in the port.
- Links denote a caller/callee relationship, ***not dataflow!***
  - e.g., linSolve port might contain: *solve(in A, out x, in b)*

A More Complex Example

- A component may *provide* and/or *use* multiple ports
- A component may *provide* some ports and *use* others
- A component may *provide* or *use* multiple instances of the same port



Ports, Interoperability, and Reuse

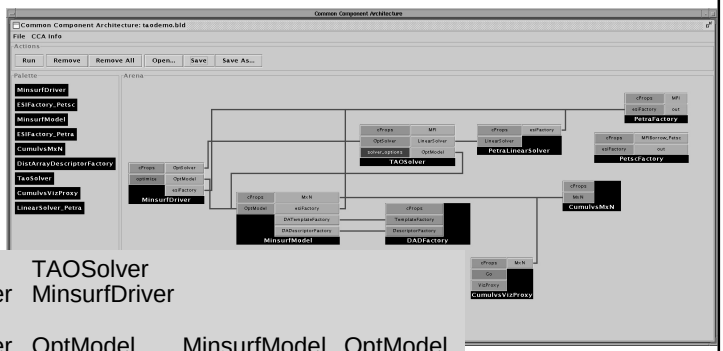
- Ports (interfaces) define how components interact
- Generality, quality, robustness of ports is up to designer/architect
  - "Any old" interface is easy to create, but…
  - Developing a robust domain "standard" interface requires thought, effort, and cooperation
- General "plug-and-play" interoperability of components requires multiple implementations conforming to the same interface
- Designing for interoperability and reuse requires "standard" interfaces
  - Typically domain-specific
  - "Standard" need not imply a formal process, may mean "widely used"

CCA Concepts: Frameworks

- The framework provides the means to "hold" components and compose them into applications
- The framework is the application's "main" or "program"
- Frameworks allow exchange ports among components without exposing implementation details
- Frameworks may support sequential, distributed, or parallel execution models, or any combination they choose
- Frameworks provide a small set of standard services to components
  - BuilderServices allow programs to compose CCA apps
- Frameworks may make themselves appear as components in order to connect to components in other frameworks

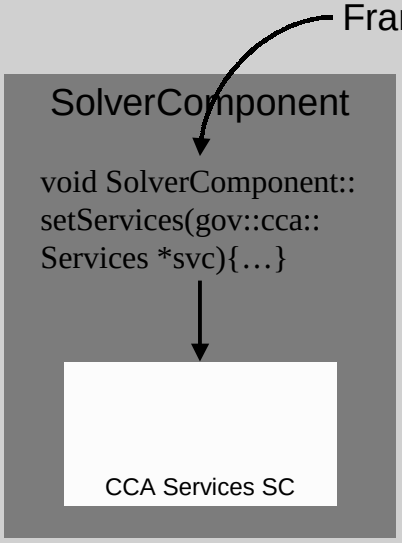
What Does This Look Like?

- Launch framework (w/ or w/o GUI)
- Instantiate components required for app.
- Connect appropriate provided and used ports
- Start application (i.e. click Go port)



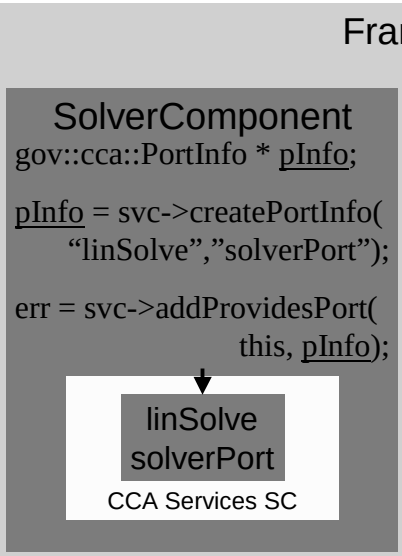

```
create   TaoSolver      TAOSolver
create   MinsurfDriver  MinsurfDriver
…
connect  MinsurfDriver  OptModel    MinsurfModel  OptModel
connect  MinsurfDriver  OptSolver   TAOSolver     OptSolver
…
```

What Makes a CCA Component?



- User instantiates ***SolverComponent***
- Framework calls ***SC***'s setServices method with a CCA Services object – the component's window on the external world

SC Provides a solverPort



- Within setServices, component declares ports it *provides* and *uses*
- addProvidesPort places port in CCA Services – makes it visible the *framework*
- Other components cannot yet use solverPort!
- setServices completes and control returns to framework to instantiate other components

PC Wants to Use a solverPort

Framework

- User instantiates ***PhysicsComponent***
- Framework calls ***PC***'s setServices
- registerUsesPort informs the *framework* that we want to connect to a component providing a solverPort
- setServices completes, control returns to framework
- PC cannot see ***SolverComponent*** or the solverPort it provides

PhysicsComponent

```
gov::cca::PortInfo * pInfo;

pInfo = svc->createPortInfo(
  "usesSolver","solverPort");

err = svc->registerUsesPort(
                        pInfo);
```

usesSolver
solverPort

CCA Services PC

User Instructs Framework to Connect solverPort between *User* and *Provider*

SolverComponent

PhysicsComponent

```
ccafe> connect  PhysicsComponent  usesSolver \
                SolverComponent   linSolve
```

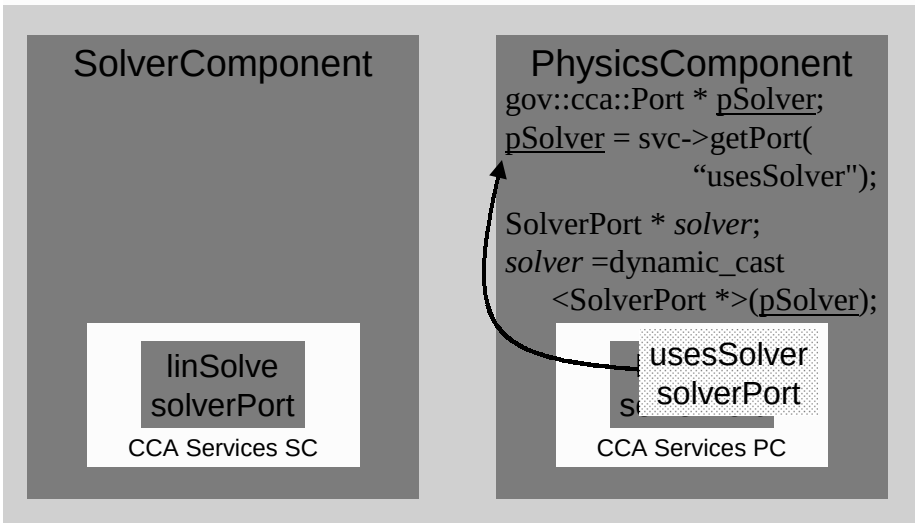
linSolve
solverPort

CCA Services SC

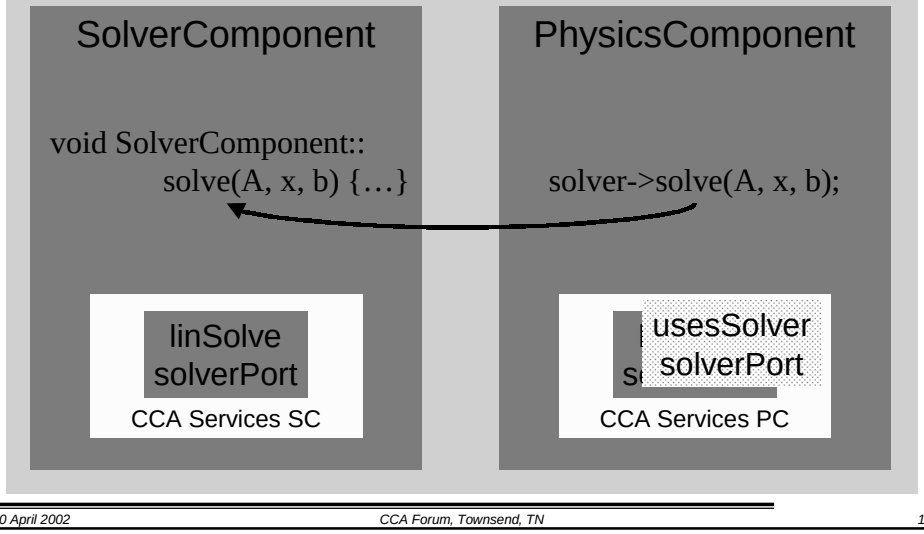
linSolve
solverPort

CCA Services PC

PC Gets the Port from its CCA Services

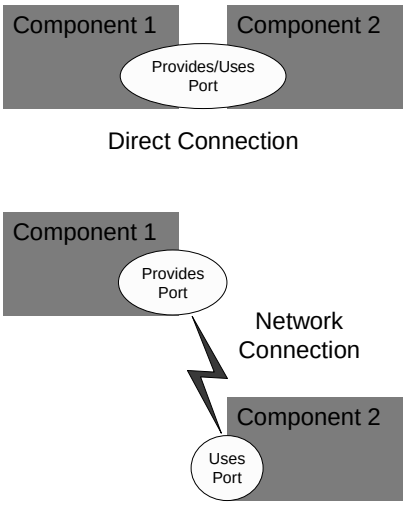


PC Can Finally Call solve on SC's solverPort



Importance of Provides/Uses Pattern for Ports

- Fences between components
  - Components must declare both what they provide and what they use
  - Components cannot interact until ports are connected
  - No mechanism to call anything not part of a port
- Ports preserve high performance direct connection semantics…
- …While also allowing distributed computing

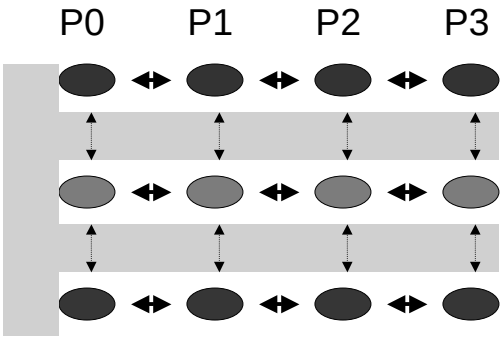


CCA Concepts: Direct Connection

- Components loaded into separate *namespaces* in same *address space* (process) from shared libraries
- getPort call returns a pointer to the port's function table
- Invoking a method on a port is equivalent to a C++ virtual function call: lookup function, invoke
- Maintains performance (lookup can be cached)

CCA Concepts: SCMD

- Single component multiple data (SCMD) model is component analog of widely used SPMD model
- Each process loaded with the same set of components wired the same way
- Different components in same process "talk to each" other via ports and the framework
- Same component in different processes talk to each other through their favorite communications layer (i.e. MPI, PVM, GA)

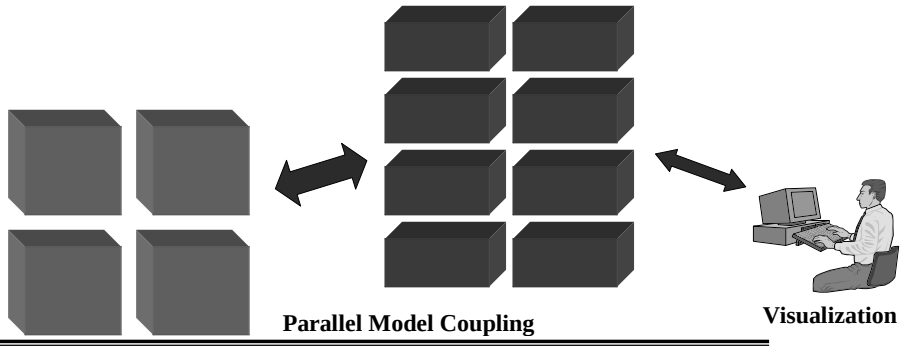


Components: Blue, Green, Red

Framework: Beige

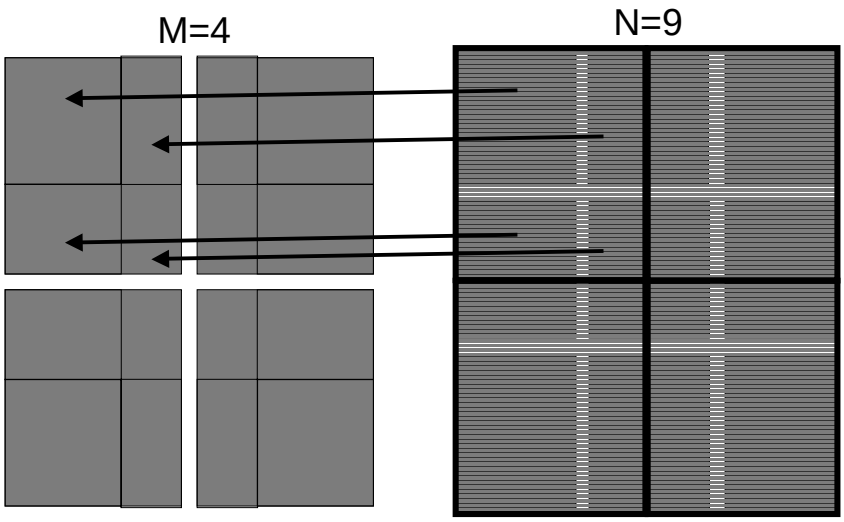
CCA Concepts: MxN Parallel Data Redistribution

- Share Data Among Coupled Parallel Models
  - Disparate Parallel Topologies (M processes vs. N)
  - e.g. Ocean & Atmosphere, Solver & Optimizer…
  - e.g. Visualization (Mx1, increasingly, MxN)



Parallel Model Coupling **Visualization**

MxN: The Problem



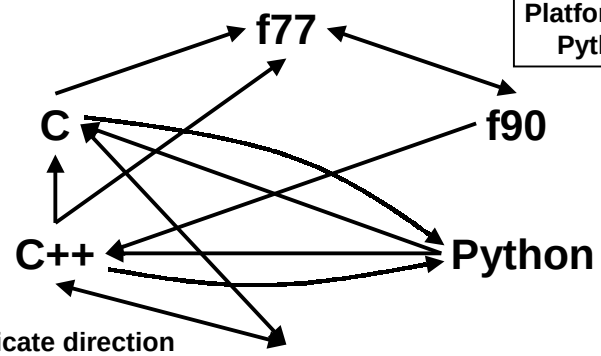
MxN Research Activities

- Distributed data descriptors
  - Uniform langauge for expressing distribution
  - Draft specification for dense multi-dimensional arrays
- MxN port
  - Draft specification completed, implemented
  - Minimal intrusion to “instrument” component, third-party control possible
  - Multiple data fields, exchange in either direction
  - One-shot or repeated transfer
- Future
  - Framework-based solutions
  - Higher-level coupling, account for units of data, spatial and termporal interpolation, etc.

CCA Concepts: Language Interoperability

- Existing language interoperability approaches are "point-to-point" solutions

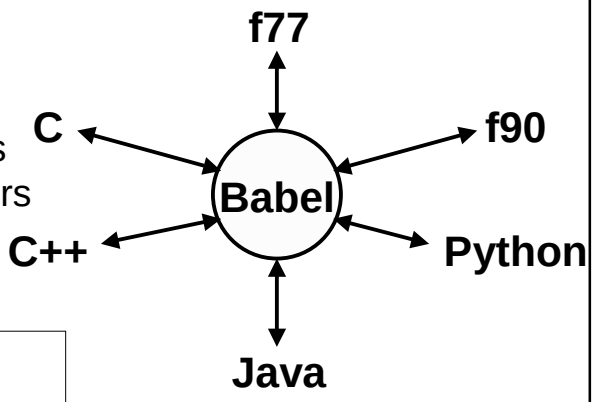
Java JNI
Native
SWIG/SILOON
Platform Dependent
Python Library



Arrows indicate direction of supported function calls

Language Interoperability w/ Babel

- Babel provides a unified approach in which all languages are considered peers

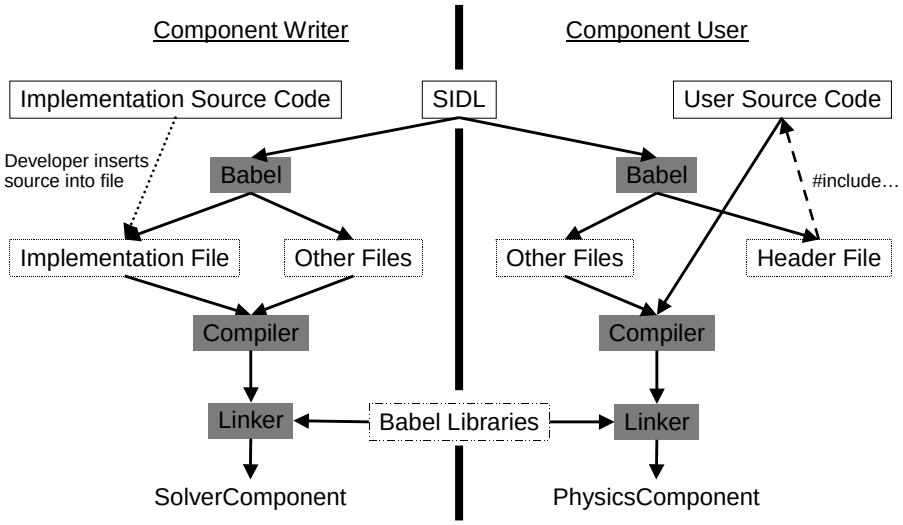


solverPort.sidl

```
version solverPort 1.0;
package solverPort {
  class Solver {
    int solve( in   array<double,2>  A,
               out array<double,1>  x,
               in   array<double, 1> b);
  }
}
```

Somewhat similar to the CORBA approach in the business domain

Using Babel



Concept Review

- Ports
  - Interfaces between components
  - Uses/provides model
- Framework
  - Allows assembly of components into applications
- Direct Connection
  - Maintain performance of local inter-component calls
- Single Component Multiple Data (SCMD)
  - Component analog of single program multiple data model
- MxN Parallel Data Redistribution
  - Model coupling, visualization, etc.
- Language Interoperability
  - Babel
  - Scientific Interface Definition Language (SIDL)

Summary

- Components promote modularity & reuse, allow developers to focus on their areas of expertise
- CCA is a component model targeted specifically to the needs of high-performance computing
  - Supports direct connection of components for local perf.
  - Supports distributed computing
  - Supports parallel computing by “staying out of the way”
- Components exchange ports following a uses-provides design pattern.
- Specification intentionally places minimal requirements on components
  - 1 additional method to become a component
  - 2 calls to declare a used or provided port
  - 2 calls required to get a port for use