CCA
Common Component Architecture

Common Component Architecture Concepts

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org



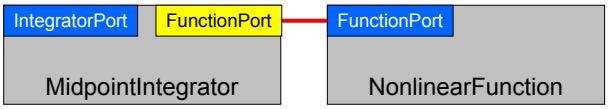
Goals

- Introduce essential features of the Common Component Architecture
- Provide common vocabulary for remainder of tutorial
- What distinguishes CCA from other component environments?

Features of the Common Component Architecture

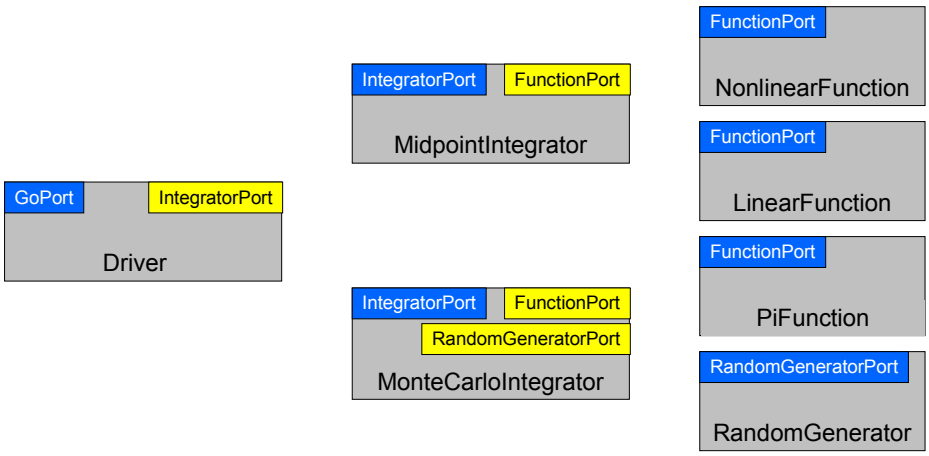
- A component model specifically designed for high-performance computing
  - Support HPC languages *(Babel)*
  - Support parallel as well as distributed execution models
  - Minimize performance overhead
- Minimalist approach makes it easier to componentize existing software
- Component interactions are *not* merely dataflow
- 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

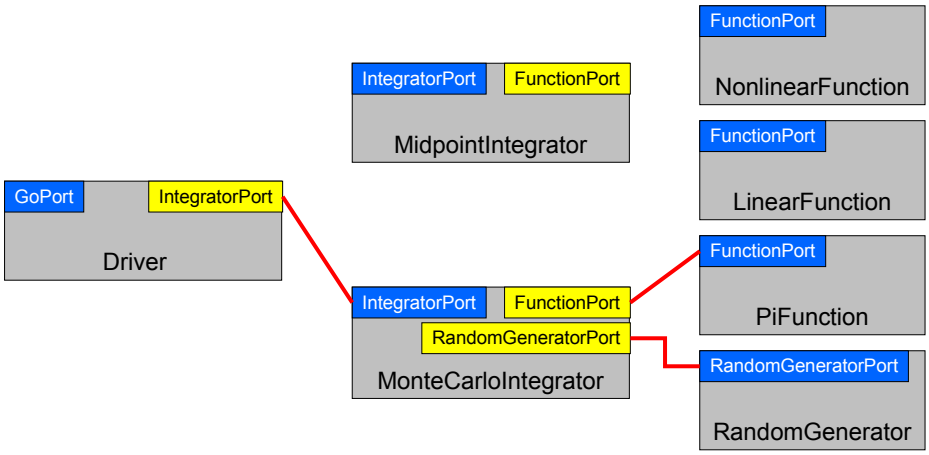


- Components interact through well-defined interfaces, or *ports*
  - In OO languages, a port is a class or interface
  - In Fortran, a port is a bunch of subroutines or a module
- Components may *provide* ports – implement the class or subroutines of the port
- Components may *use* ports – call methods or subroutines in the port
- Links denote a caller/callee relationship, ***not dataflow!***
  - e.g., FunctionPort could contain: *evaluate(in Arg, out Result)*

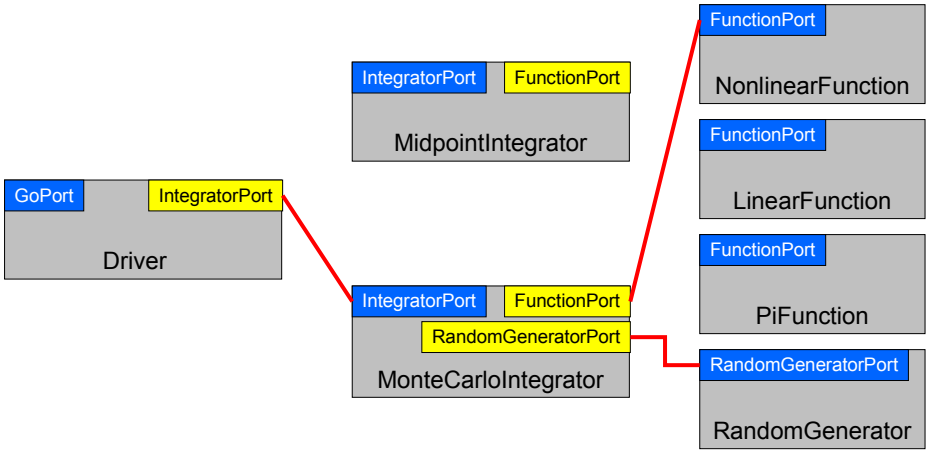
Components and Ports in the Integrator Example



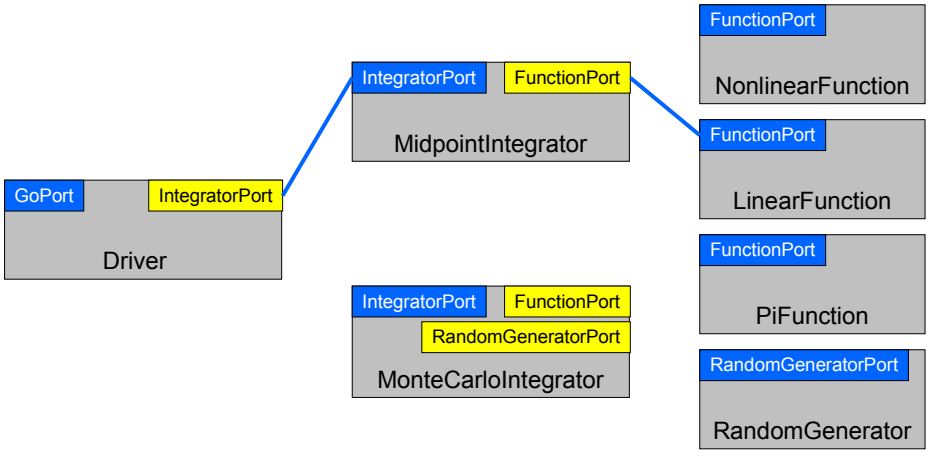
An Application Built from the Example Components



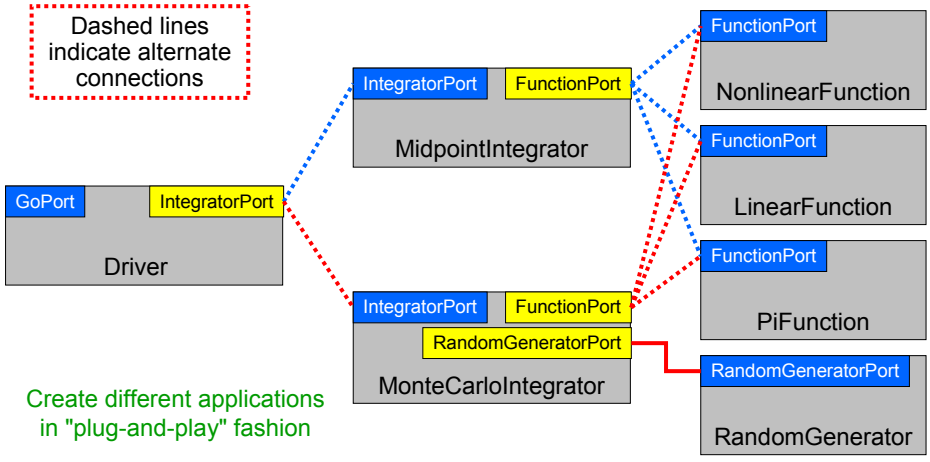
Another Application…



Application 3…



And Many More…

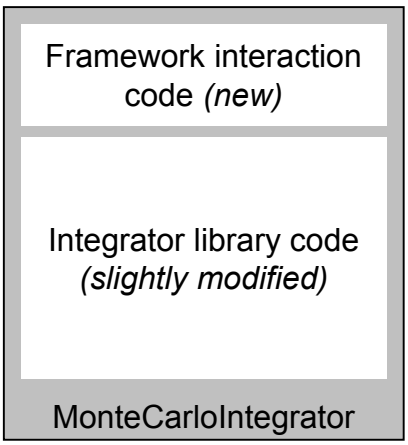


Ports, Interoperability, and Reuse

- 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"

Components vs Libraries

- Component environments rigorously enforce interfaces
- Can have several versions of a component loaded into a single application
- Component needs add'l code to interact w/ framework
  - Constructor and destructor methods
  - Tell framework what ports it *uses* and *provides*
- Invoking methods on other components requires slight modification to "library" code



CCA Concepts: Frameworks

- The framework provides the means to "hold" components and compose them into applications
  - The framework is often application's "main" or "program"
- Frameworks allow exchange of ports among components without exposing implementation details
- 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
- *Currently:* specific frameworks support specific computing models (parallel, distributed, etc.). *Future:* full flexibility through integration or interoperation

The Lifecycle of a Component

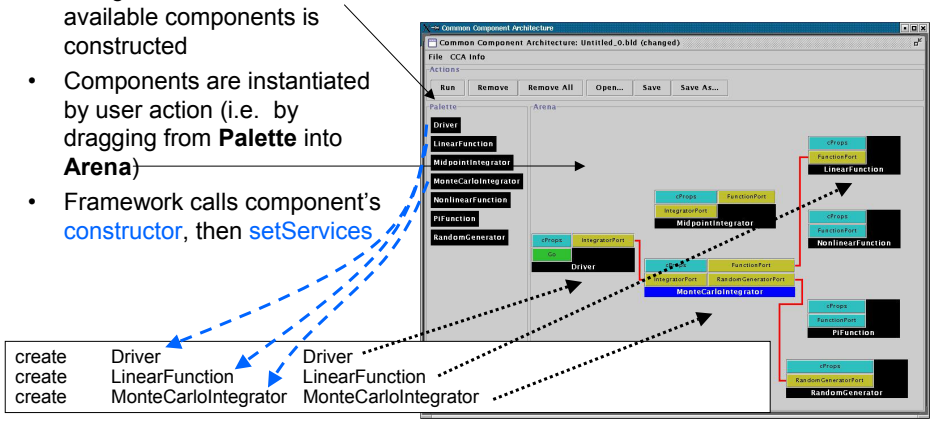
- User instructs framework to load and *instantiate* components
- User instructs framework to connect *uses* ports to *provides* ports
- Code in components uses functions provided by another component
- Ports may be disconnected
- Component may be destroyed

Look at actual code in next tutorial module

Loading and Instantiating Components

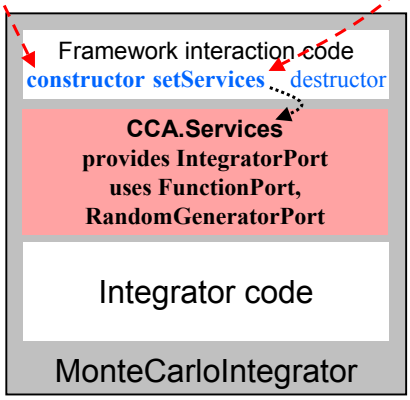
- Components are code (usu. library or shared object) + metadata
- Using metadata, a **Palette** of available components is constructed
- Components are instantiated by user action (i.e. by dragging from **Palette** into **Arena**)
- Framework calls component's constructor, then setServices

- Details are framework-specific!
- Ccaffeine currently provides both command line and GUI approaches

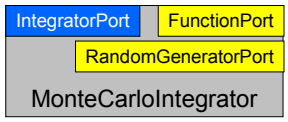


Component's View of Instantiation

- Framework calls component's constructor
- Component initializes internal data, etc.
  - Knows *nothing* outside itself

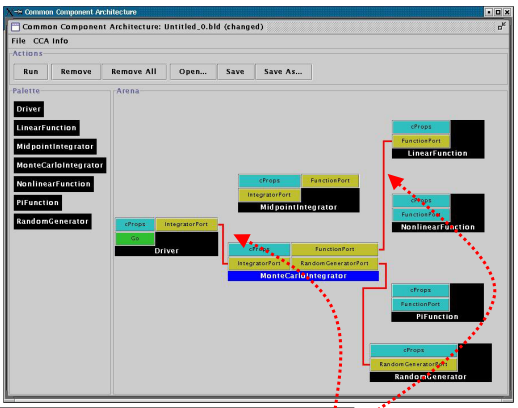


- Framework calls component's setServices
  - Passes setServices an object representing everything "outside"
  - setServices declares ports component *uses* and *provides*
- Component *still* knows nothing outside itself
  - But Services object provides the means of communication w/ framework
- Framework now knows how to "decorate" component and how it might connect with others



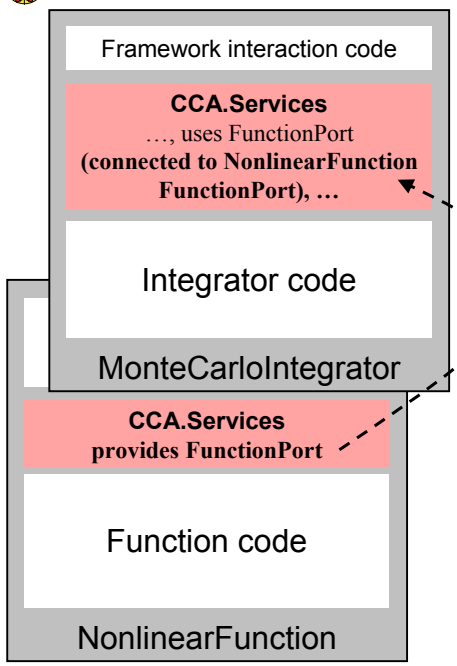
User Connects Ports

- Can only connect user & provider
  - Not uses/uses or provides/provides
- Ports connected by type, not name
  - Port names must be unique within component
  - Types must match across components
- Framework puts info about *provider* into *user* component's Services object



connect	Driver	IntegratorPort	MonteCarloIntegrator	IntegratorPort
connect	MonteCarloIntegrator	FunctionPort	LinearFunction	FunctionPort
…				

Component's View of Connection

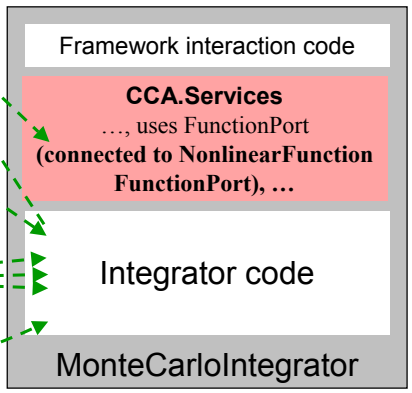


- Framework puts info about provider into user component's Services object
  - ***MonteCarloIntegrator***'s Services object is aware of connection
  - ***NonlinearFunction*** is not!
- ***MCI***'s integrator code cannot yet call functions on FunctionPort

Component's View of Using a Port

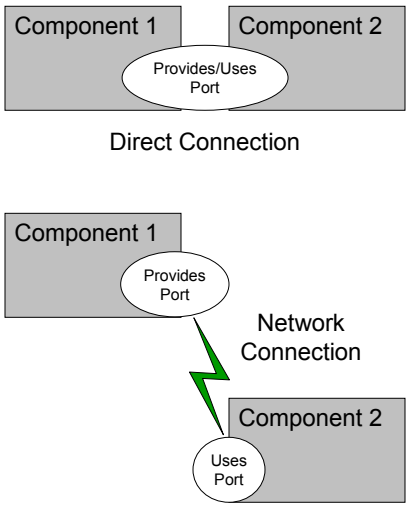
- User calls getPort to obtain (handle for) port from Services
  - Finally user code can "see" provider
- Cast port to expected type
  - OO programming concept
  - Insures type safety
  - Helps enforce declared interface
- Call methods on port
  - e.g.

  sum = sum + function->evaluate(x)
- Release port



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

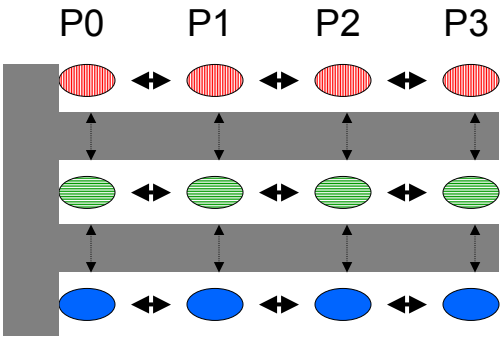


CCA Concepts: Direct Connection

- Components loaded into separate namespaces in the same address space (process) from shared libraries
- getPort call returns a pointer to the port's function table
- Calls between components equivalent to a C++ virtual function call: lookup function location, invoke
- Cost equivalent of ~2.8 F77 or C function calls
- All this happens "automatically" – user just sees high performance
- *Description reflects Ccaffeine implementation, but similar or identical mechanisms in other direct connect fwks*

CCA Concepts: Parallel Components

- 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)
- Also supports MPMD/MCMD



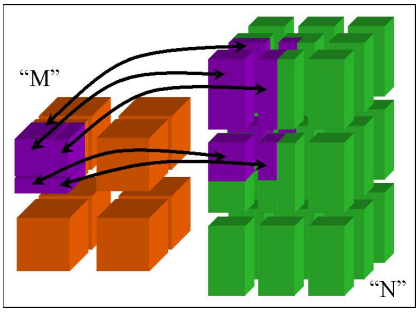
Components: Red, Green, Blue

Framework: Gray

Framework stays "out of the way" of component parallelism

CCA Concepts: MxN Parallel Data Redistribution

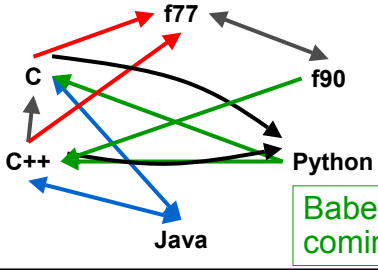
- Share Data Among Coupled Parallel Models
  - Disparate Parallel Topologies (M processes vs. N)
  - e.g. Ocean & Atmosphere, Solver & Optimizer…
  - e.g. Visualization (Mx1, increasingly, MxN)



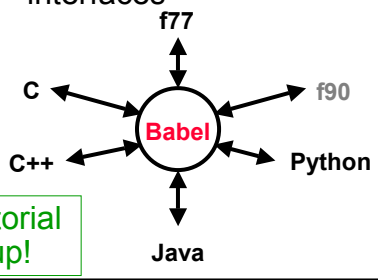
Research area -- tools under development

CCA Concepts: Language Interoperability

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



- Babel provides a unified approach in which all languages are considered peers
- Babel used primarily at interfaces



Babel tutorial coming up!

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
- Parallelism
  - Framework stays out of the way of parallel components
- MxN Parallel Data Redistribution
  - Model coupling, visualization, etc.
- Language Interoperability
  - Babel, Scientific Interface Definition Language (SIDL)

Next: A Simple CCA Example