CCA
Common Component Architecture

Common Component Architecture Concepts

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

CCA
Common Component Architecture

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

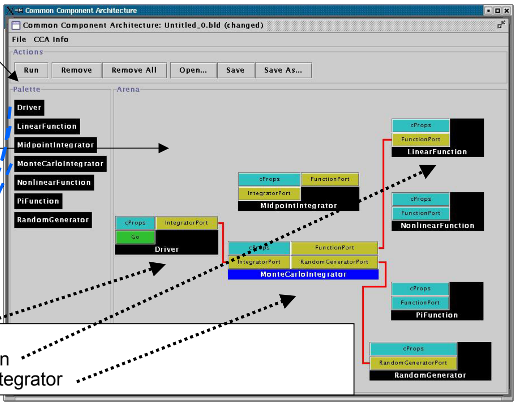
Look at actual code in next tutorial module

CCA Common Component Architecture

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

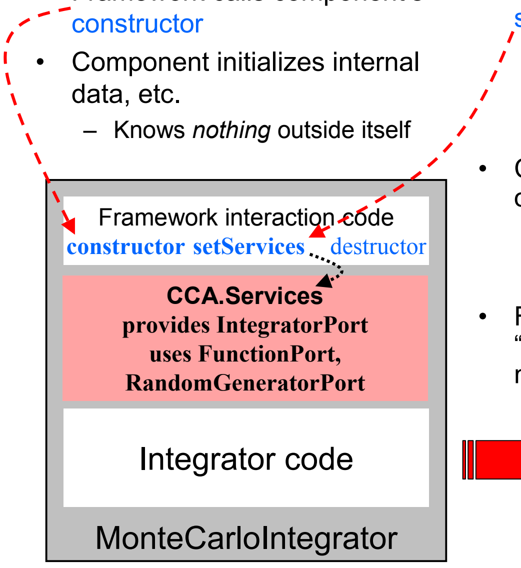


```
create    Driver                  Driver
create    LinearFunction          LinearFunction
create    MonteCarloIntegrator    MonteCarloIntegrator
```

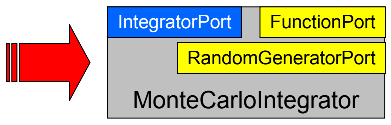
CCA Common Component Architecture

Component's View of Instantiation

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


Framework interaction code
constructor setServices destructor
CCA.Services
provides IntegratorPort
uses FunctionPort, RandomGeneratorPort
Integrator code
MonteCarloIntegrator

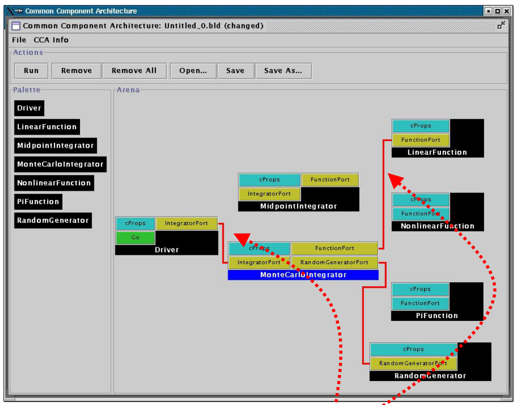
- Framework calls component's setServices
  - Passes setServices an object representing everything "outside"
  - setServices declares ports component *uses* and *provides*
- Component *still* knows nothing outside itself
  - But Services object provides the means of communication w/ framework
- Framework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



CCA
Common Component Architecture

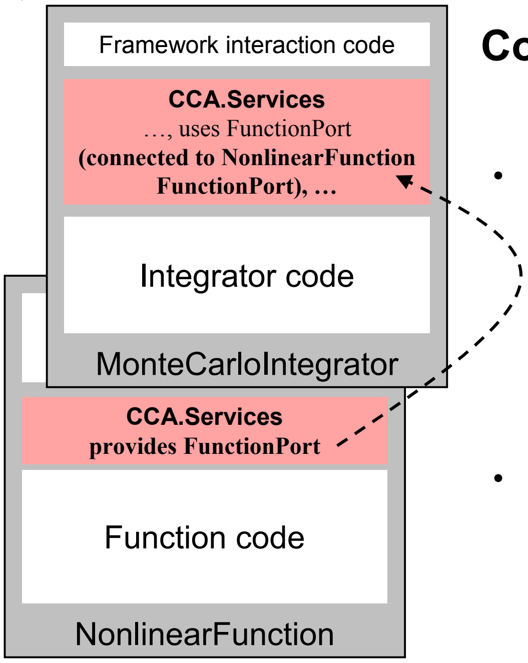
User Connects Ports

- 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
…
```

CCA
Common Component Architecture



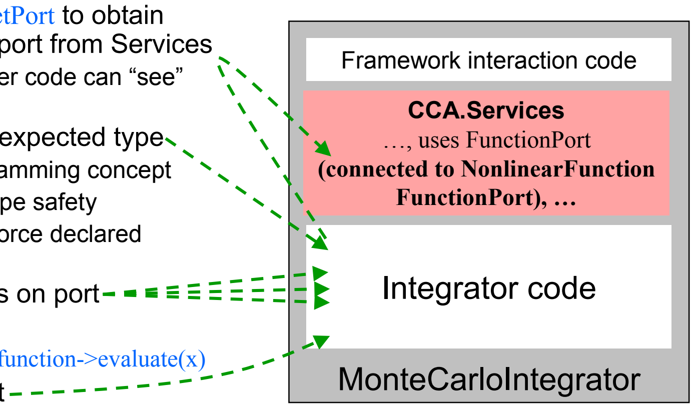
Component's View of Connection

- 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

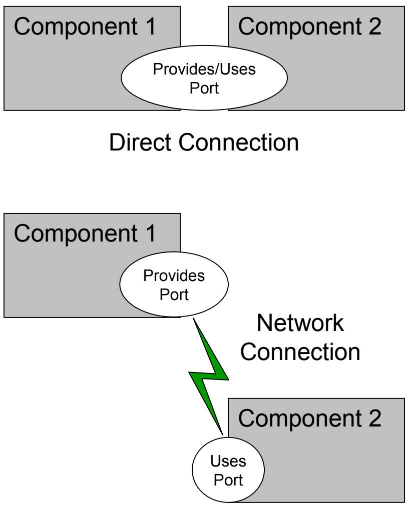
- User calls getPort to obtain (handle for) port from Services
  - Finally user code can "see" provider
- Cast port to expected type
  - OO programming concept
  - Insures type safety
  - Helps enforce declared interface
- Call methods on port
  - e.g.

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


Framework interaction code
CCA.Services
…, uses FunctionPort
(connected to NonlinearFunction FunctionPort), …
Integrator code
MonteCarloIntegrator

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
Common Component Architecture

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
Common Component Architecture

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