CCA
Common Component Architecture

Common Component Architecture Concepts

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

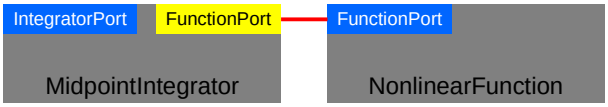

Goals

- 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

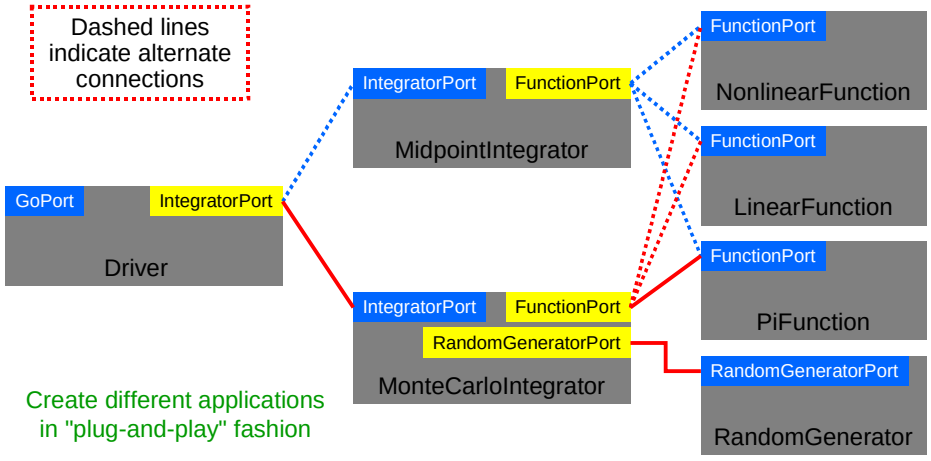
- A component model specifically designed for high-performance computing
  - Support HPC languages *(Babel)*
  - Support parallel as well as distributed execution models
  - Minimize performance overhead
- Minimalist approach makes it easier to componentize existing software
- Component interactions 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)*

Ports in the Integrator Example



Ports, Interoperability, and Reuse

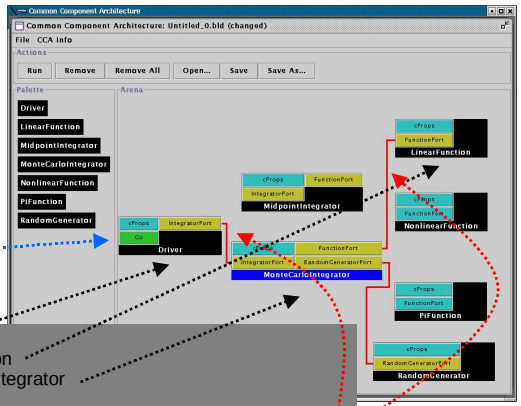
- Ports (interfaces) define how components interact
- Generality, quality, robustness of ports is up to designer/architect
  - "Any old" interface is easy to create, but…
  - Developing a robust domain "standard" interface requires thought, effort, and cooperation
- General "plug-and-play" interoperability 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 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

User Interaction w/ Framework

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



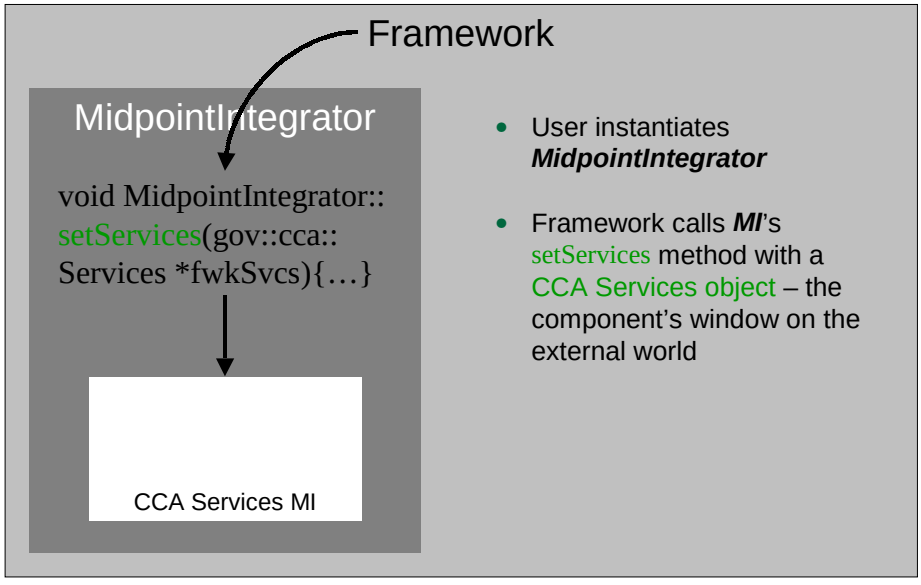

```
create      Driver                Driver
create      LinearFunction        LinearFunction
create      MonteCarloIntegrator  MonteCarloIntegrator
…
connect     Driver                IntegratorPort  MonteCarloIntegrator  IntegratorPort
connect     MonteCarloIntegrator  FunctionPort    LinearFunction        FunctionPort
…
```

Interactions Between Components and Frameworks

- When component is instantiated…
  - Framework calls component's setServices
  - setServices registers ports to be used or provided
- When user connects a *uses* port to a *provides* port…
  - CCA Services object in *uses* component "becomes aware" of provider's implementation
- When component wants to use a port…
  - Get a pointer to the port with getPort *(once)*
  - Call methods on the port *(many times)*

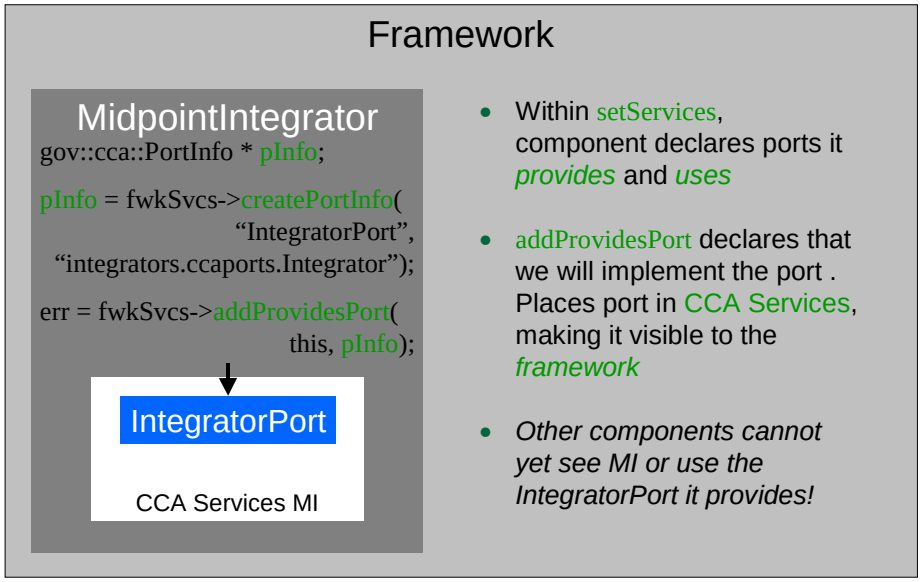
Look at actual code in next tutorial module

What Makes a CCA Component?



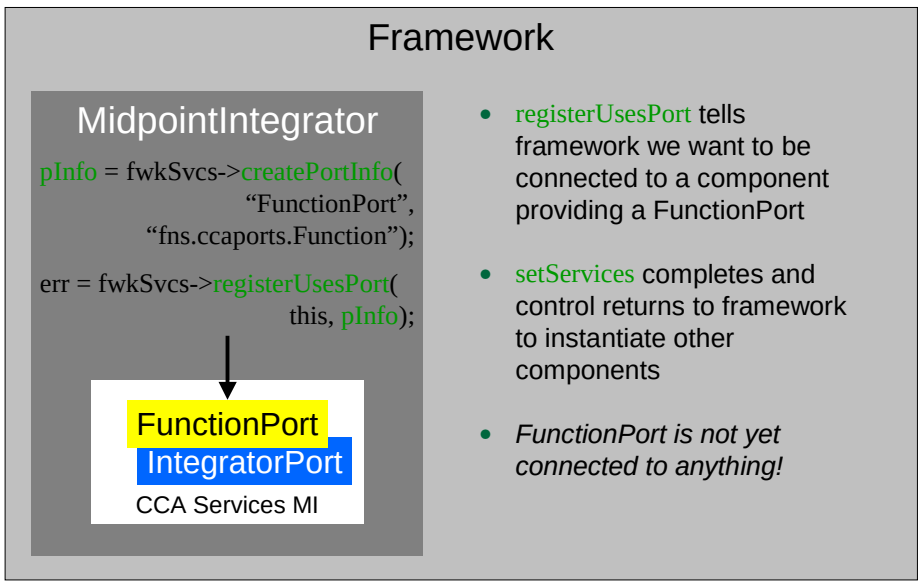
- User instantiates ***MidpointIntegrator***
- Framework calls ***MI***'s setServices method with a CCA Services object – the component's window on the external world

MI Provides an IntegratorPort…



- Within setServices, component declares ports it *provides* and *uses*
- addProvidesPort declares that we will implement the port . Places port in CCA Services, making it visible to the *framework*
- *Other components cannot yet see MI or use the IntegratorPort it provides!*

…and Uses a FunctionPort



- registerUsesPort tells framework we want to be connected to a component providing a FunctionPort
- setServices completes and control returns to framework to instantiate other components
- *FunctionPort is not yet connected to anything!*

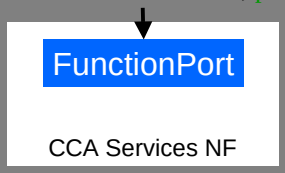
NF Provides a FunctionPort

Framework

- User instantiates ***NonlinearFunction***
- Framework calls ***NF***'s setServices
- addProvidesPort informs the *framework* that we implement a FunctionPort
- setServices completes, control returns to framework
- *MI cannot yet see* ***NonlinearFunction*** *or the FunctionPort it provides*

NonlinearFunction

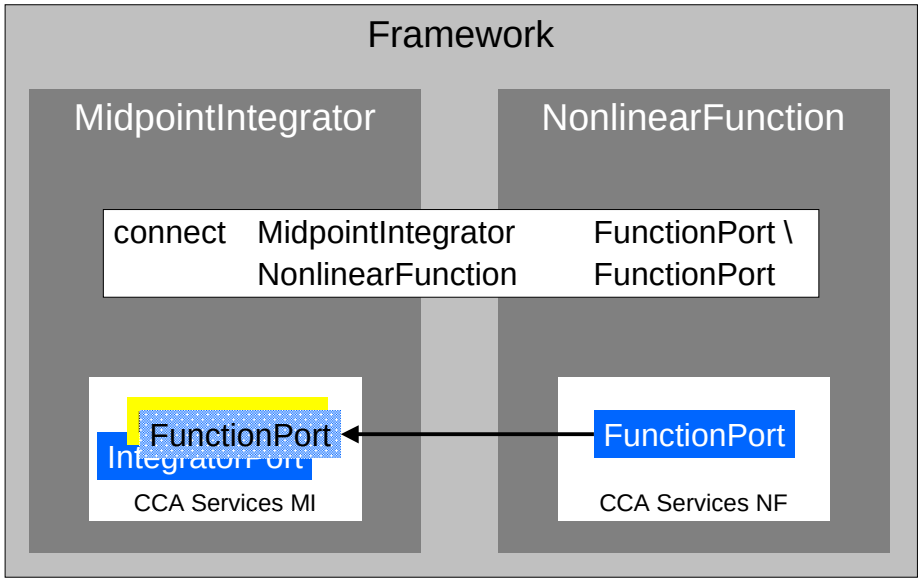
```
gov::cca::PortInfo * pInfo;

pInfo = fwkSvcs->createPortInfo(
                    "FunctionPort",
                "fns.ccaports.Function");

err = fwkSvcs->addProvidesPort(
                        this, pInfo);
```



Interactions Between Components and Frameworks

- When component is instantiated…
  - Framework calls component's setServices
  - setServices registers ports to be used or provided
- When user connects a *uses* port to a *provides* port…
  - CCA Services object in *uses* component "becomes aware" of provider's implementation
- When component wants to use a port…
  - Get a pointer to the port with getPort *(once)*
  - Call methods on the port *(many times)*

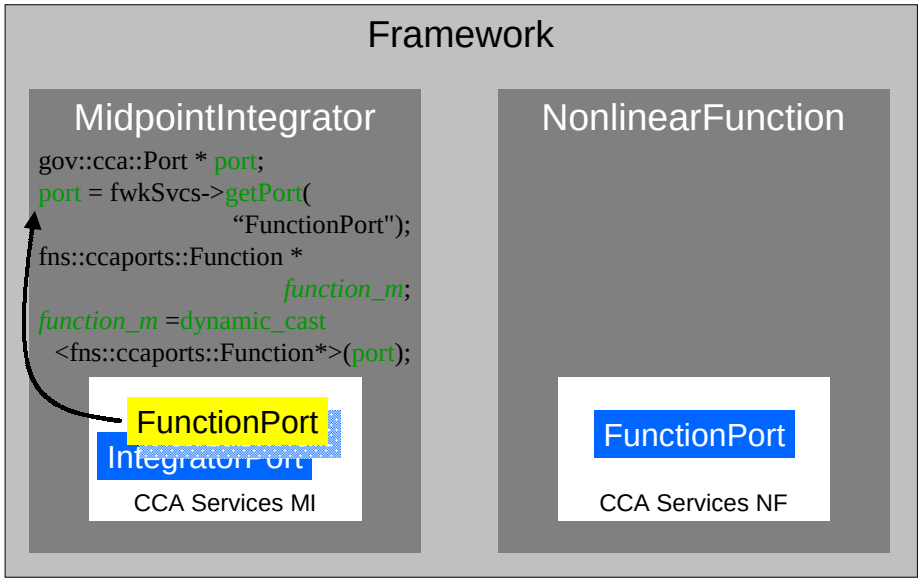
User Tells Framework to Connect Ports



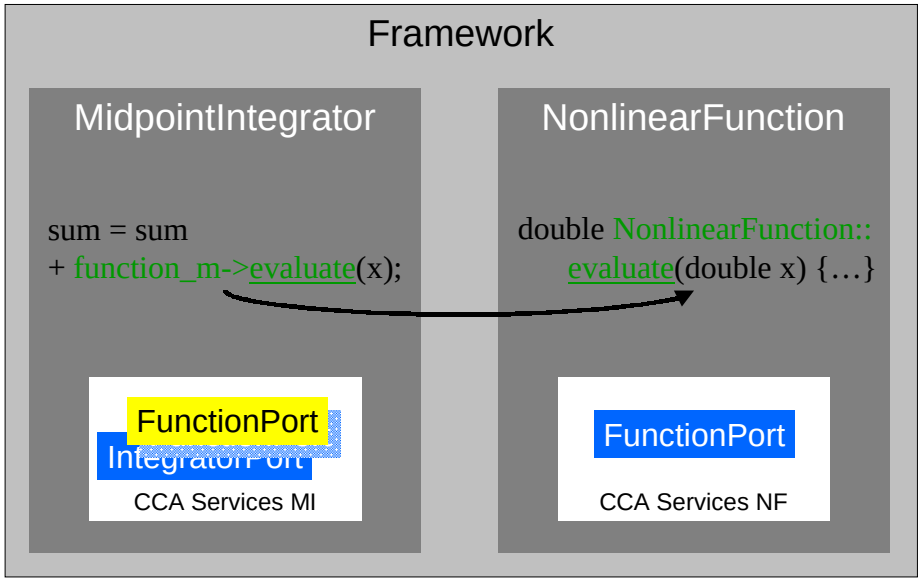
Interactions Between Components and Frameworks

- When component is instantiated…
  - Framework calls component's setServices
  - setServices registers ports to be used or provided
- When user connects a *uses* port to a *provides* port…
  - CCA Services object in *uses* component "becomes aware" of provider's implementation
- When component wants to use a port…
  - Get a pointer to the port with getPort *(once)*
  - Call methods on the port *(many times)*

MI Gets Port from its CCA Services

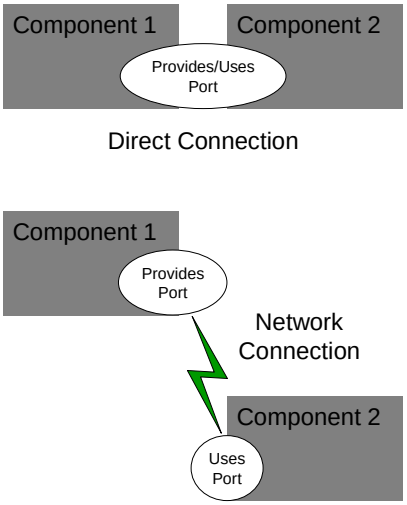


MI Gets Port from its CCA Services



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

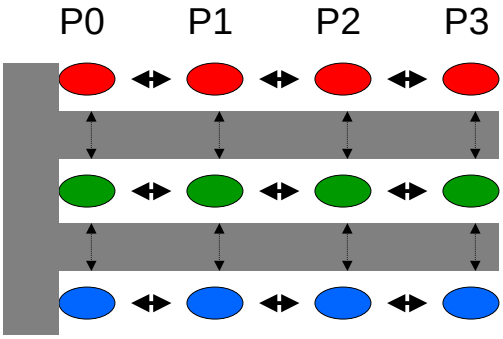


CCA Concepts: Direct Connection

- 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



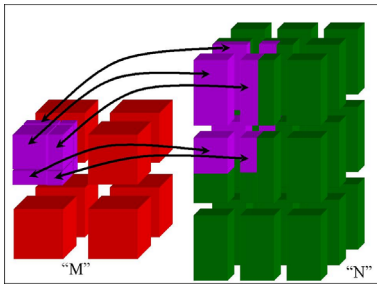
Components: Red, Green, Blue

Framework: Gray

Framework stays "out of the way" of component parallelism

CCA Concepts: MxN Parallel Data Redistribution

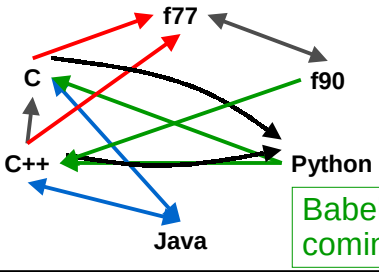
- Share Data Among Coupled Parallel Models
  - Disparate Parallel Topologies (M processes vs. N)
  - e.g. Ocean & Atmosphere, Solver & Optimizer…
  - e.g. Visualization (Mx1, increasingly, MxN)



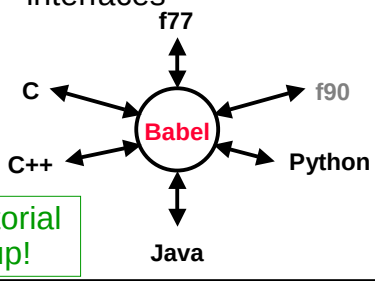
Research area -- tools under development

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)