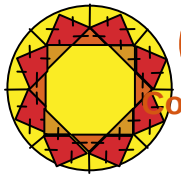


CCA
Common Component Architecture

Progress in Rapid, Sustainable Development for Complex HPC Software

19 May 2008

Benjamin Allan &

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

Who We Are: The Common Component Architecture (CCA) Forum

- Combination of standards body and user group for the CCA
- Define specifications for high-performance scientific components & frameworks
- Promote development of HPC tools for component-based software development, components, and component applications
- Open membership, quarterly meetings…

General mailing list: *cca-forum@cca-forum.org*

Web: http://www.cca-forum.org/

- Center for Technology for Advanced Scientific Component Software (TASCS)
  - Funded by the US DOE SciDAC program
  - Core development team for CCA technologies

Our HPC software tools help HPC code...

- Technical leaders to manage change and to enforce programming discipline.
- Managers and teams to work across complex organizational barriers.
- Developers to cross programming language and build system barriers.

Leaders Managing Code Complexity

Some Common Situations:

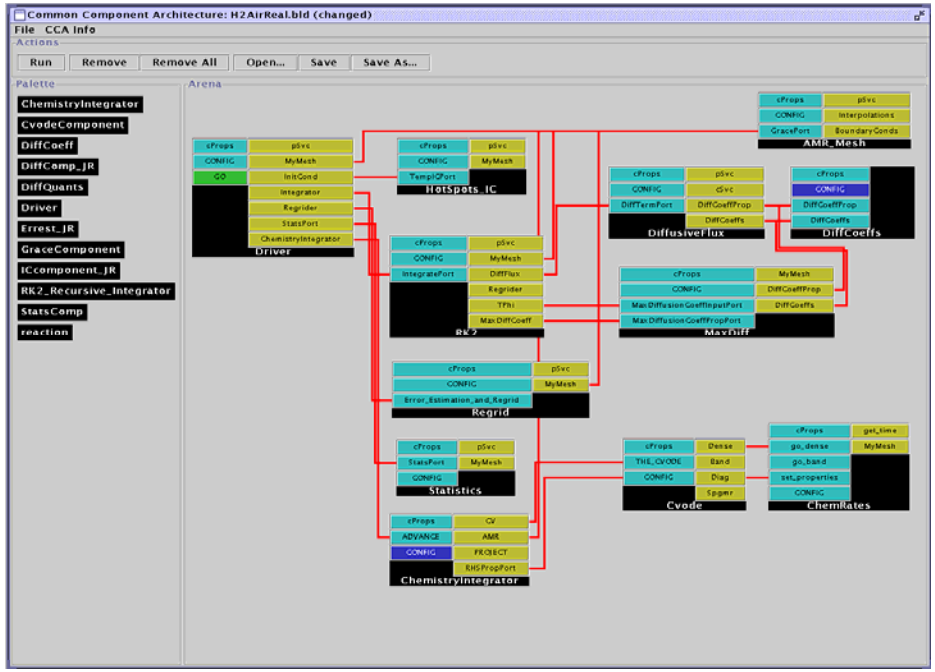
- Your code is so large and complex it has become fragile and hard to keep running
- You have a simple code, and you want to extend its capabilities – rationally
- You want to develop a modular "toolkit" that
  - Can be assembled in different ways to perform different scientific calculations
  - Gives users without programming experience access to a flexible tool for simulation
  - Gives users without HPC experience use of HPC-ready software

How CCA Can Help:

- Components help you think about software in manageable chunks that interact only in well-defined ways
- Components provide a "plug-and-play" environment that allows easy, flexible application assembly

Example: Computational Facility for Reacting Flow Science (CFRFS)

- A toolkit to perform simulations of unsteady flames
- Solve the Navier-Stokes with detailed chemistry
  - Any mechanism
  - Structured adaptive mesh refinement
- CFRFS today:
  - 100+ components
  - 9 external libraries
  - 13 contributors



"Wiring diagram" for a typical CFRFS simulation, utilizing 12 components.

CCA tools reused: Ccaffeine framework and GUI
Languages: C, C++, F77

Teams working across organizations

Some Common Situations:

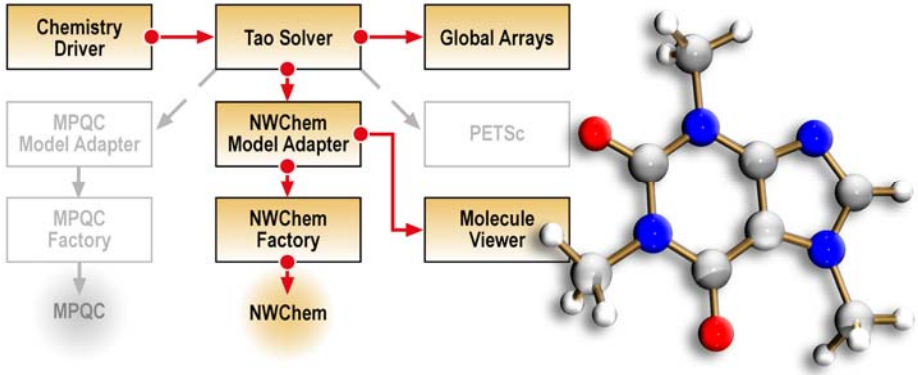
- Many (geographically distributed) developers creating a large software system.
  - Hard to coordinate, different parts of the software don't work together as required.
- Groups of developers with different expertise.
- Build communities to standardize interfaces and share code.

How CCA Can Help:

- Components are natural units for
  - Expressing software architecture
  - Encapsulating particular expertise
  - Separating the *interface* from the *implementation*

Example: Quantum Chemistry

- Integrated state-of-the-art optimization technology into two quantum chemistry packages to explore effectiveness in chemistry applications
- Geographically distributed expertise:
  - California - chemistry
  - Illinois - optimization
  - Washington – chemistry, parallel data management
- Effective collaboration with minimal face-to-face interaction



Schematic of CCA-based molecular structure determination quantum chemistry application.

Components based on: MPQC, NWChem (quantum chem.), TAO (optimization), Global Arrays, PETSc (parallel linear algebra)
CCA tools used: Babel, Ccaffeine framework and Gui
Languages: C, C++, F77, Python

Developers using multiple languages

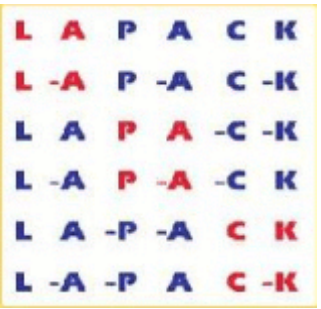
Some Common Situations:

- Need to use existing libraries written in multiple languages in the same application.
- Want to allow others to access your library from multiple languages.

How CCA Can Help:

- SIDL (language interoperability standard) and Babel (a SIDL compiler). Every language can call every other.
- Describing components and interfaces with SIDL focuses the design discussion away from the 'favorite language' wars.

Examples



hypre

- High performance preconditioners and linear solvers
- Library written in C
- Babel-generated object-oriented interfaces provided in C, C++, Fortran

LAPACK07

- Update to LAPACK linear algebra library
  - To be released 2007
  - Library written in F77, F95
- Will use Babel-generated interfaces for: C, C++, F77, F95, Java, Python
- Possibly also ScaLAPACK (distributed version)

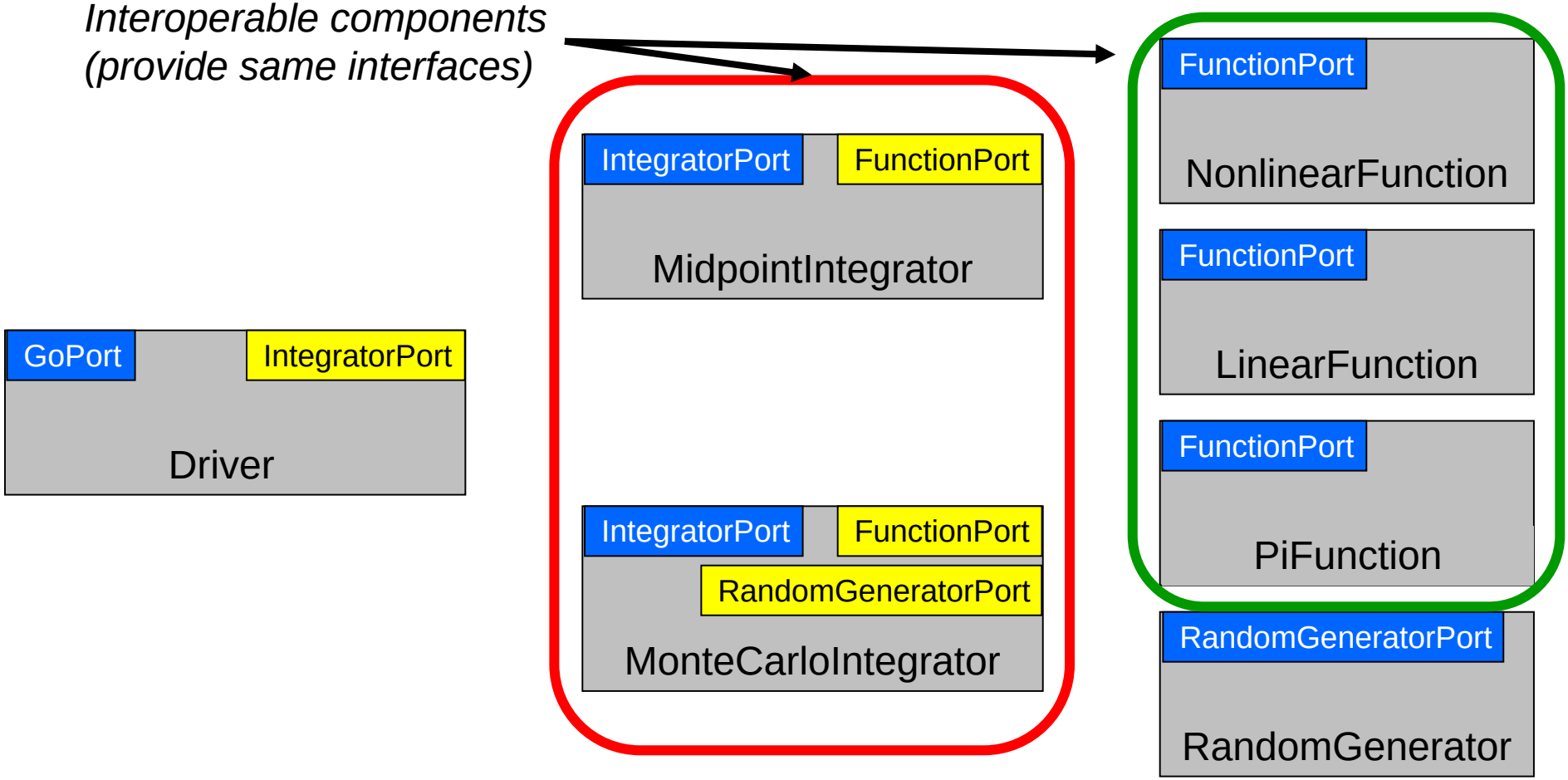
"I implemented a Babel-based interface for the hypre library of linear equation solvers. The Babel interface was straightforward to write and gave us interfaces to several languages for less effort than it would take to interface to a single language."

-- Jeff Painter, LLNL. 2 June 2003

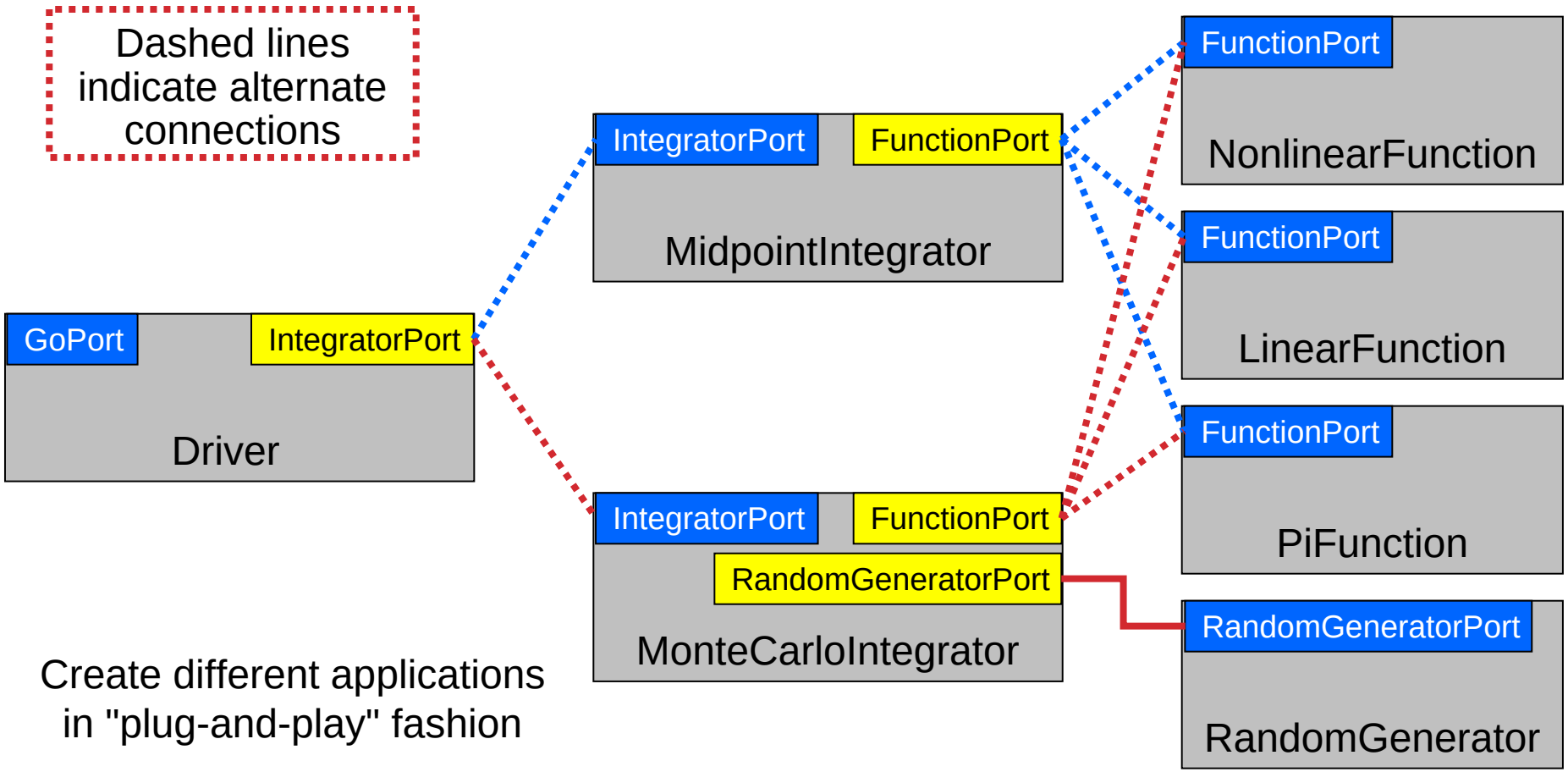
CCA tools used: Babel, Chasm

A Simple Example: Numerical Integration Components

Interoperable components (provide same interfaces)



And Many More…



Create different applications in "plug-and-play" fashion

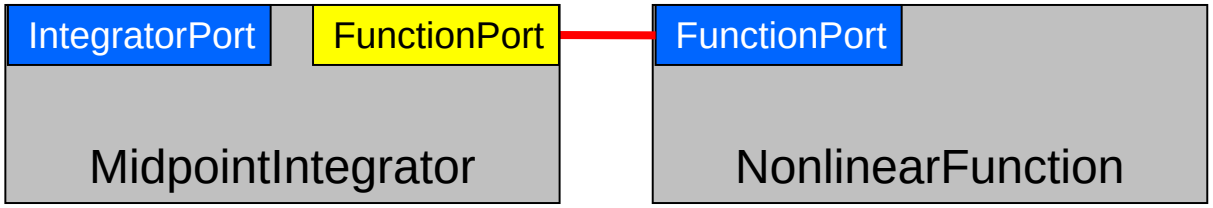
Special Needs of Scientific HPC

- Support for legacy software
  - How much change required for component environment?
- Performance is important
  - What overheads are imposed by the component environment?
- Both parallel and distributed computing are important
  - What approaches does the component model support?
  - What constraints are imposed?
  - What are the performance costs?
- Support for languages, data types, and platforms
  - Fortran?
  - Complex numbers? Arrays? (as first-class objects)
  - Is it available on my parallel computer?

CCA: Concept and Practice

- In the following slides, we explain important concepts of component-based software from the CCA perspective
- We also sketch how these concepts are manifested in code (full details in the Hands-On)
- The CCA Specification is the mapping between concept and code
  - A standard established by the CCA Forum
  - Expressed in the Scientific Interface Definition Language (SIDL) for language neutrality (syntax similar to Java)
  - SIDL can be translated into bindings for specific programming languages using, e.g., the Babel language interoperability tool

CCA Concept: Ports



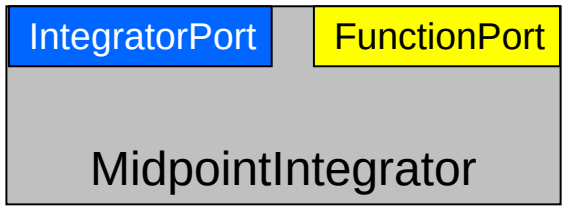
- Components interact through well-defined interfaces, or *ports*
  - A port expresses some computational functionality
  - In Fortran, a port is a bunch of subroutines or a module
  - In OO languages, a port is an abstract class or interface
- Ports and connections between them are a procedural (caller/callee) relationship, not dataflow!
  - e.g., *`FunctionPort`* could contain a method like `evaluate(in Arg, out Result)` with data flowing both ways

Components and Ports (in SIDL)

```
package gov.cca {
  interface Component {
    void setServices(…);
} }
```

```
package gov.cca {
  interface Port {
} }
```

```
package integrators {
  interface IntegratorPort
       extends gov.cca.Port
{
    double integrate(…};
} }
```



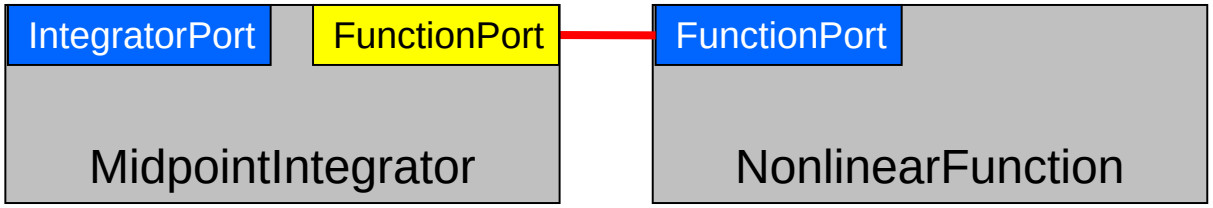

```
package integrators {
  class Midpoint implements
       gov.cca.Component,
       integrator.IntegratorPort
{
  double integrate(…);
  void setServices(…);
} }
```

Key:

= Inheritance

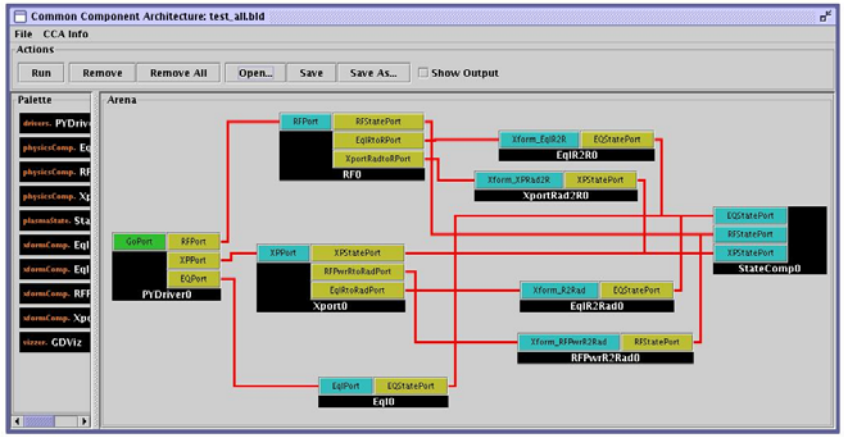
SIDL inheritance keywords

Using Ports



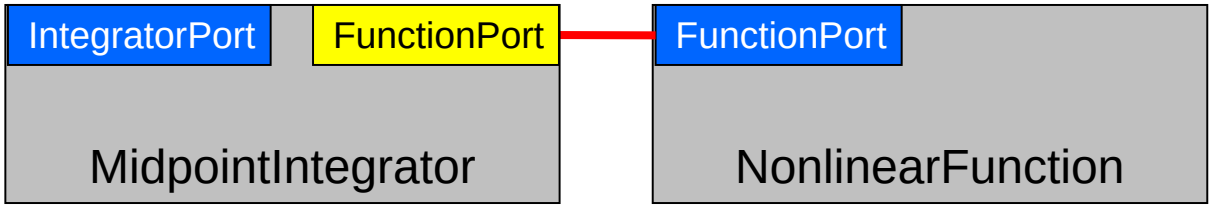
- Calling methods on a port you use requires that you first obtain a “handle” for the port
  - Done by invoking **`getPort()`** on the user’s **`gov.cca.Services`** object
  - Free up handle by invoking **`releasePort()`** when done with port
- Best practice is to bracket actual port usage as closely as possible without using **`getPort()`**, **`releasePort()`** too frequently
  - Get/Release may be expensive operations, especially in distributed computing contexts
  - Performance is in tension with dynamism
    - can’t “re-wire” a ports that is “in use”

CCA Concepts: Frameworks



- The framework is the tool that holds the components and composes them into applications at user's direction.
- Frameworks allow connection of ports without exposing component implementation details
- Frameworks provide a small set of standard services to components
  - Framework services are CCA ports, just like on components
  - Components can register ports as services using the `ServiceProvider` port
- *Currently:* framework implementations are specialized for specific computing models (parallel, distributed, etc.)
- *Future:* interoperability of frameworks

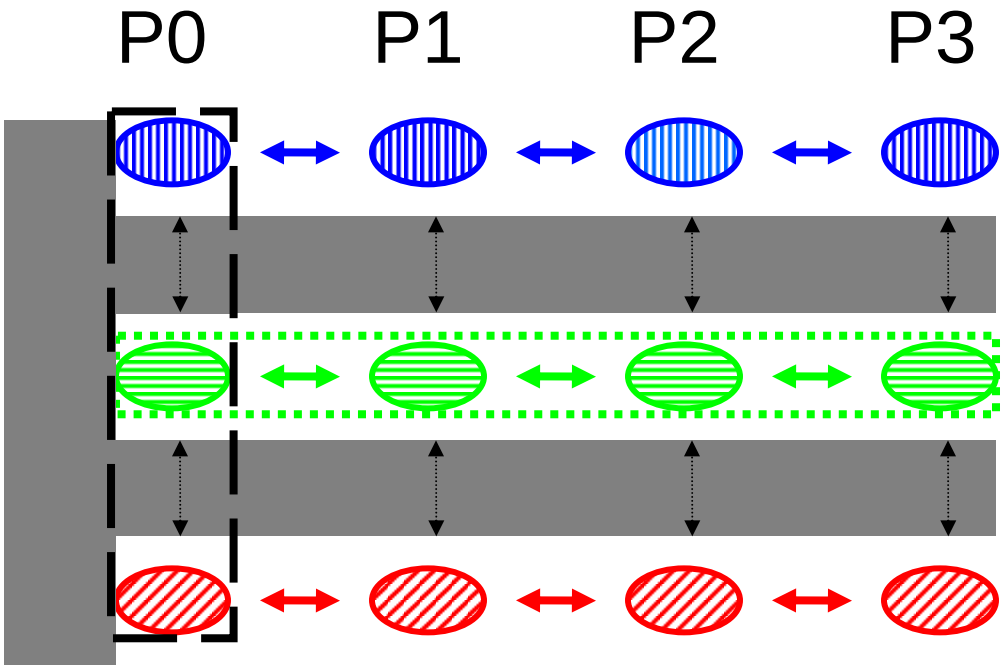
Components Must Keep Frameworks Informed



- Components must tell the framework about the ports they are providing and using
  - Framework will not allow connections to ports it isn't aware of
- Register them using methods on the component's **`gov.cca.Services`** object
  - **`addProvidesPort()`** and **`removeProvidesPort()`**
  - **`registerUsesPort()`** and **`unregisterUsesPort()`**
  - All are defined in the CCA specification
- Ports are usually registered in the component's **`setServices()`** method
  - Can also be added/removed dynamically at runtime.

CCA Supports Parallelism -- by "Staying Out of the Way" of it

- 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: Gray

Any parallel programming environments that can be mixed outside of CCA can be mixed inside

Maintaining HPC Performance

- The performance of your application is as important to us as it is to you

More about performance in notes

- The CCA is designed to provide maximum performance
  - But the best we can do is to make your code perform no worse, *unless we give easy access to new algorithms*.
- Facts:
  - Measured overheads per function call are low
  - Most overheads easily amortized by doing enough work per call
  - Other changes made during componentization may also have performance impacts
  - *Awareness* of costs of abstraction and language interoperability improves designs for high performance

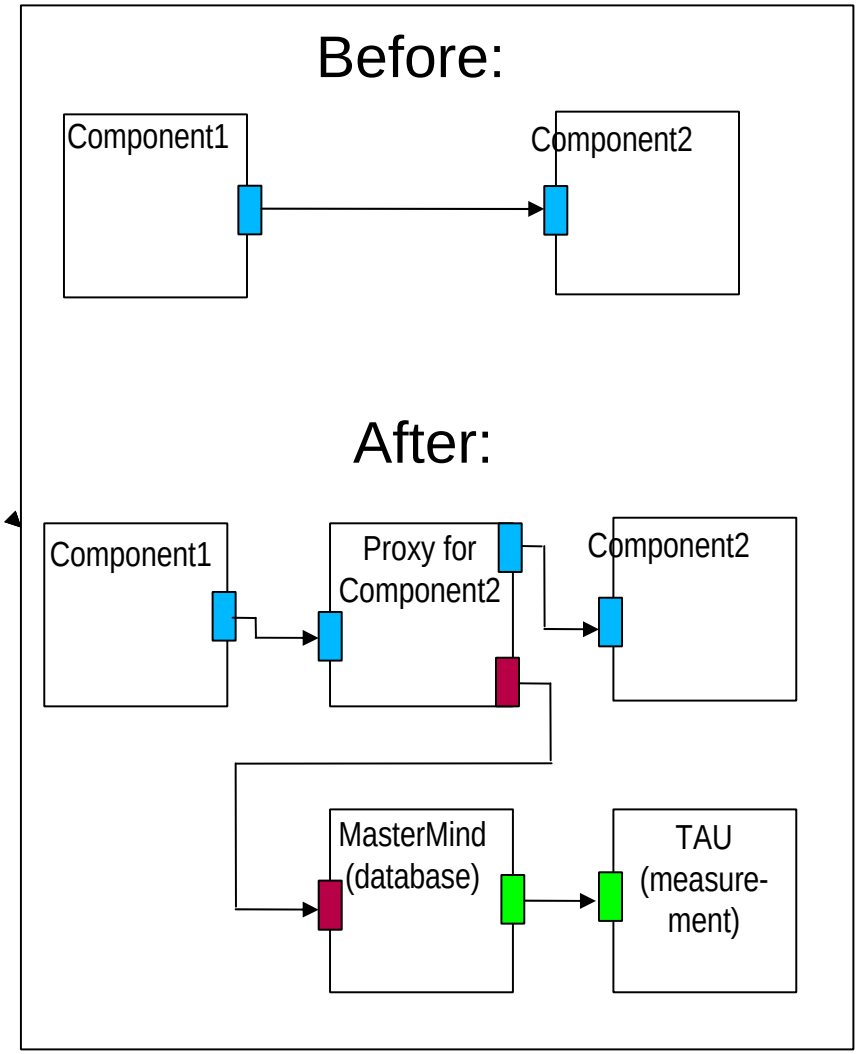
The Proxy Component Pattern

- A "proxy" component can be inserted between the user and provider of a port without either being aware of it (non-invasive)
- Proxy can observe or act on all invocations of the interface
- For many purposes, proxies can be generated automatically from SIDL definition of the port

Sample uses for proxy components:

- Performance: instrumentation of method calls
- Debugging: execution tracing, watching data values
- Testing: Capture/replay

Performance Monitoring with TAU



Component Lifecycle

Additional material in notes

- **Composition Phase (assembling application)**
  - Component is instantiated in framework
  - Component interfaces are connected appropriately
- **Execution Phase (running application)**
  - Code in components uses functions provided by another component
- **Decomposition Phase (termination of application)**
  - Connections between component interfaces may be broken
  - Component may be destroyed

In an application, individual components may be in different phases at different times

Steps may be under human or software control

Is CCA for You?

- Much of what CCA does can be done without such tools *if* you have sufficient discipline
  - The larger a group, the harder it becomes to impose the necessary discipline
- Projects may use different aspects of the CCA
  - CCA is *not* monolithic – use what *you* need
  - Few projects use all features of the CCA… at first.
- Evaluate what *your* project needs against CCA's capabilities
  - Other groups' criteria probably differ from yours
  - CCA continues to evolve, so earlier evaluations may be out of date
- Evaluate CCA against other ways of obtaining the desired capabilities
- Suggested starting point:
  - CCA tutorial "hands-on" exercises

Take an Evolutionary Approach

- The CCA is designed to allow selective use and incremental adoption
- "SIDLize" interfaces incrementally
  - Start with essential interfaces
  - Remember, only externally exposed interfaces need to be in SIDL.
- Componentize at successively finer granularities
  - Start with whole application as one component
    - Basic feel for components without "ripping apart" your app.
  - Subdivide into finer-grain components as appropriate
    - Code reuse opportunities
    - Plans for code evolution

The tools

- Bocca – project environment
- Ccaffeine – framework
- SIDL – interoperability language
- Babel – HPC language binding generator
- CCA – specification for components, frameworks

Bocca Development Environment

- Provides a text-based, portable environment
  - Create or import SIDL and CCA based codes.
  - Automatic build system maintenance.
  - Easy to adopt or abandon while preserving code, build.
- No GUI required.
- Still in the early beta stage of development
  - Being tested by managing the tutorial source and a regression test suite.
  - Basis for common CCA toolkit installation.
  - Manages components in all Babel-supported languages (C, C++, Fortran 77, Fortran 90, Java, Python).

Bocca Creates Skeletons for CCA

- Including ports and interfaces
  - Give the SIDL name and an empty port or interface is created.
- Including components and classes
  - Give the name and an empty component or class is created.
  - Some extra options: the component uses/provides ports, implemented interfaces or extended classes
- Including the build system
  - For all ports/components in the project
  - Implemented in any CCA supported language
- Create applications with Ccaffeine GUI
- Including application composition (coming soon)

Bocca Example

```
# create an empty but buildable CCA skeleton
bocca create project myproj
cd myproj
./configure

bocca create port myJob
bocca create component myWorker –provides=myJob:job1

# fill in public functionality
bocca edit port myJob

# fill in implementation
bocca edit component –i myWorker

make
```

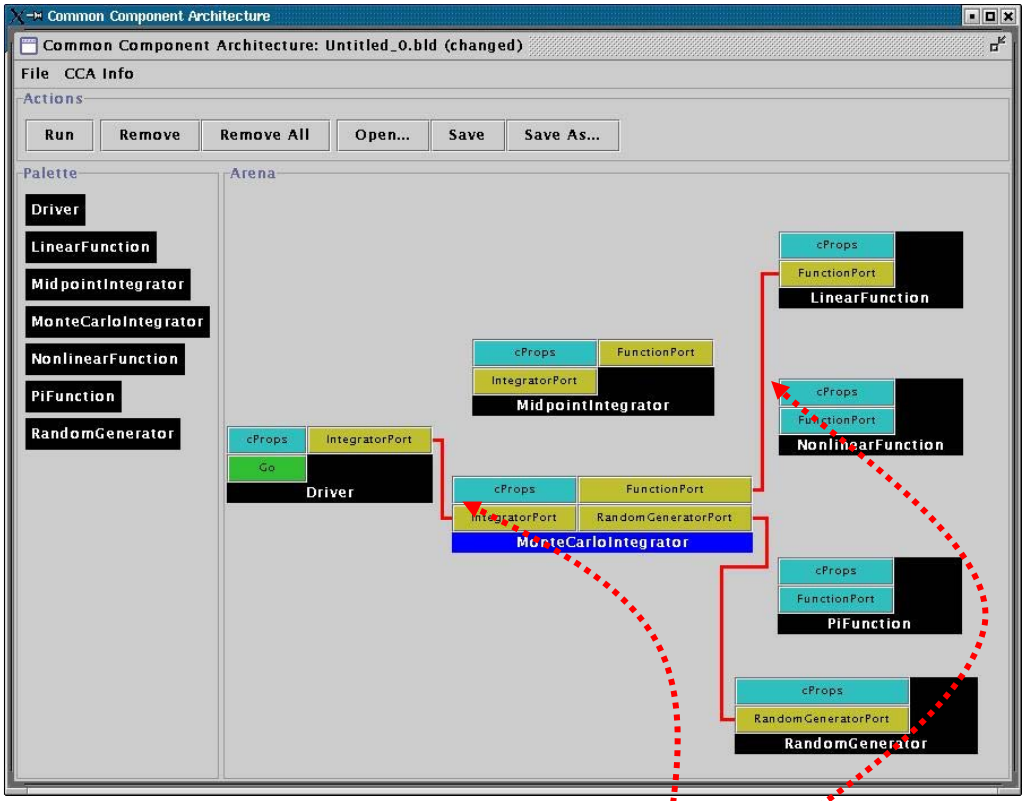
Ccaffeine is a *Direct-Connect*, Parallel-Friendly Framework

- Supports SIDL/Babel components
  - Conforms to latest CCA specification (0.8)
  - Also supports legacy CCA specification (0.5)
    - Any C++ allowed with C and Fortran by C++ wrappers
- Provides command-line and GUI for composition
  - Scripting supports batch mode for SPMD
  - MPMD/SPMD custom drivers in any Babel language

Supported on Linux, AIX, OSX and is portable to modern UNIXes.

User Connects Ports

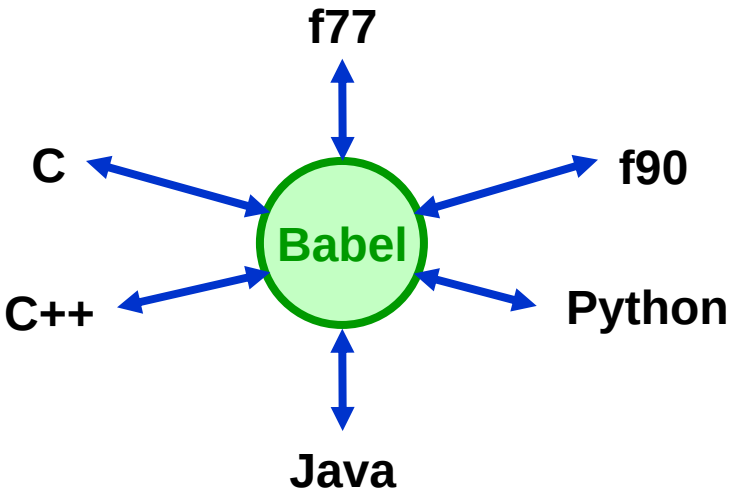
- Can only connect caller(right) and callee(left)
- Ports connected by type
  - Port instance names are unique within a component
  - Port types match across a connection
- Framework puts info about *provider* of port into the *using component's* Services handle



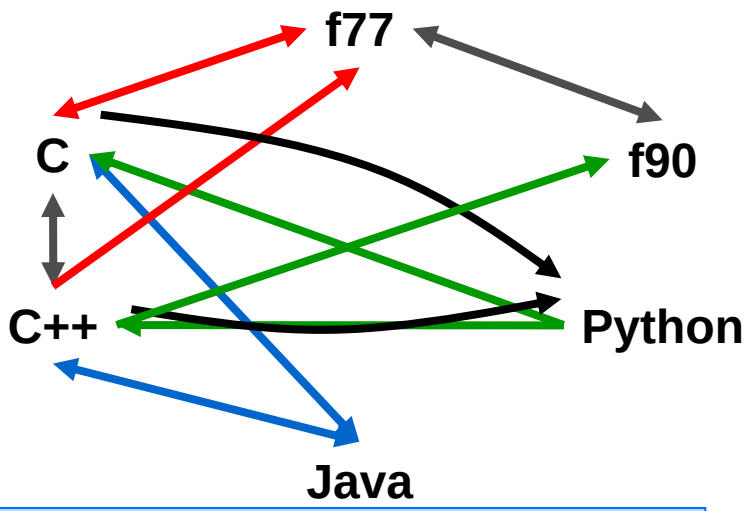
connect	Driver	IntegratorPort	MonteCarloIntegrator	IntegratorPort
connect	MonteCarloIntegrator	FunctionPort	LinearFunction	FunctionPort
…				

SIDL Facilitates *Scientific* Programming Language Interoperability

- Programming language-neutral interface descriptions
- Native support for basic scientific data types
  - Complex numbers
  - Multi-dimensional, multi-strided arrays
- Automatic object-oriented wrapper generation
- Usable standalone or in CCA environment



vs.



Supported on Linux, AIX, works on OSX, catamount;
C (ANSI C), C++ (GCC), F77 (g77, Sun f77), F90 (Intel, Lahey, GNU, Absoft, PGI), Java (1.4)

The SIDL File that defines the "greetings.English" type

①② `package greetings version 1.0 {`

③④ `  interface Hello {`

⑤ `    void setName( in string name );`

`    string sayIt ( );`

`  }`

⑥ `  class English implements-all Hello { }`

`}`

Working Code: "Hello World" in F90 Using a Babel Type

```
program helloclient
  use greetings_English
  use sidl_BaseInterface
  implicit none
  type(greetings_English_t) :: obj
  type(sidl_BaseInterface_t):: exc
  character (len=80)        :: msg
  character (len=20)        :: name
  name='World'
  call new( obj, exc )
  call setName( obj, name, exc )
  call sayIt( obj, msg, exc )
  call deleteRef( obj, exc )
  print *, msg
end program helloclient
```

Looks like a native F90 derived type

These subroutines were specified in the SIDL.

Other basic subroutines are "built in" to all Babel types.

Question: What language is "obj" really implemented in?

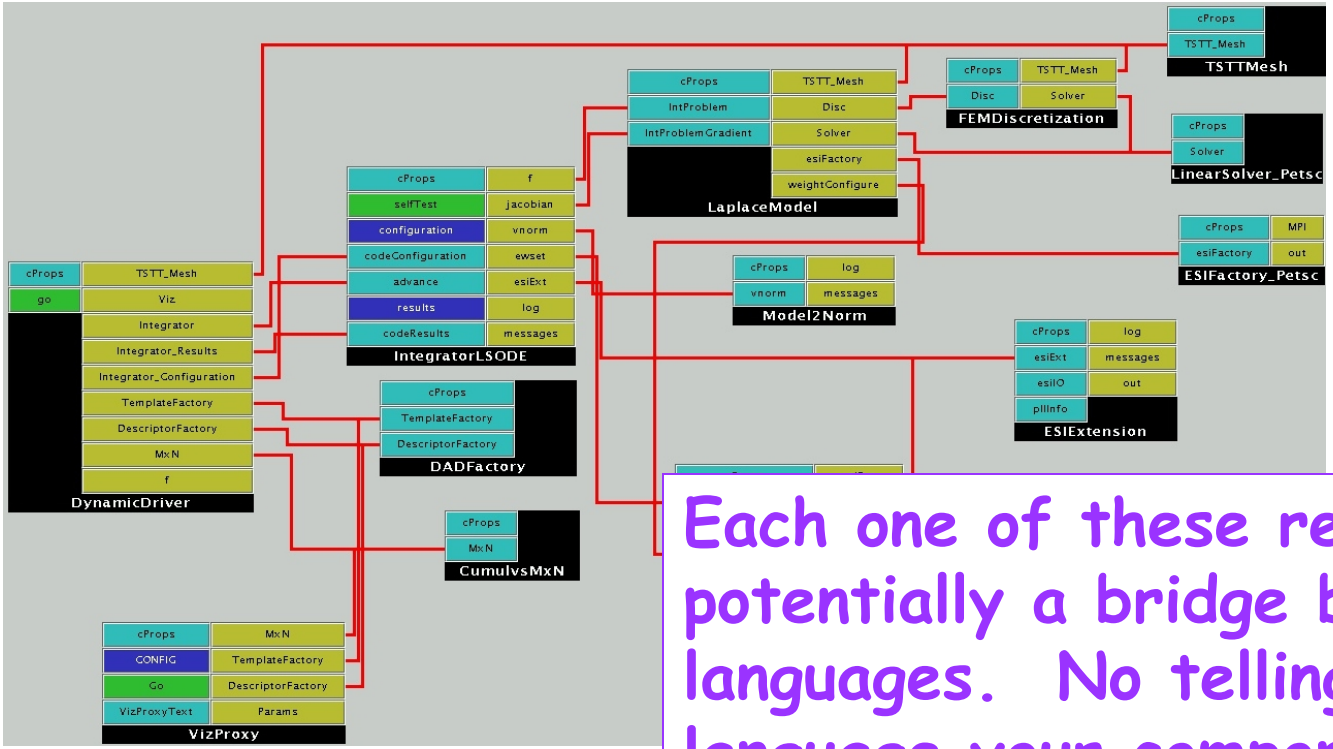
```
program helloclient
  use greetings_English
  use sidl_BaseInterface
  implicit none
  type(greetings_English_t) :: obj
  type(sidl_BaseInterface_t):: exc
  character (len=80)        :: msg
  character (len=20)        :: name
  name='World'
  call new( obj, exc )
  call setName( obj, name, exc )
  call sayIt( obj, msg, exc )
  call deleteRef( obj, exc )
  print *, msg
end program helloclient
```

Answer: Can't Know!

With Babel, it could be C, C++, Python, Java, Fortran77, or Fortran90/95

In fact, it could change on different runs without recompiling this code!

CCA uses Babel for high-performance n-way language interoperabilty



Each one of these red lines, is potentially a bridge between two languages. No telling which language your component will be connected to when you write it.

Implementation Details Must be Filled in Between Splicer Blocks

```
namespace greetings {
class English_impl {
  private:
    // DO-NOT-DELETE splicer.begin(greetings.English._impl)
    string d_name;
    // DO-NOT-DELETE splicer.end(greetings.English._impl)
```

```
string
greetings::English_impl::sayIt()
throw ()
{
  // DO-NOT-DELETE splicer.begin(greetings.English.sayIt)
  string msg("Hello ");
  return msg + d_name + "!";
  // DO-NOT-DELETE splicer.end(greetings.English.sayIt)
}
```

Resources: Its All Online

- Information about all CCA tutorials, past, present, and future:
  http://www.cca-forum.org/tutorials/

- Specifically…
  - Latest versions of hands-on materials and code:
    http://www.cca-forum.org/tutorials/#sources
    - Hands-On designed for self-study as well as use in an organized tutorial
    - Should work on most Linux distributions, less tested on other unixen
    - Still evolving, so please contact us if you have questions or problems
  - Archives of all tutorial presentations:
    http://www.cca-forum.org/tutorials/archives/

- Questions…
  help@cca-forum.org or cca-tutorial@cca-forum.org

What else is in development?

- Toolkits of numerics, meshes, & other basics.
- Automatic wrapping tools for C/C++/Fortran (generate the SIDL and implementation from your non-CCA primary sources).
- Comprehensive documentation.
- Support for programming-by-contract.