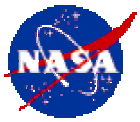
CCA
Common Component Architecture

Welcome to the Common Component Architecture Tutorial

SC2003

17 November, 2003



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

CCA
Common Component Architecture

Agenda & Table of Contents

Time	Title	Slide No.	Presenter
8:30-8:35am	Welcome	1	David Bernholdt, ORNL
8:35-9:15am	Introduction to Components	8	David Bernholdt, ORNL
9:15-10:00am	CCA Concepts	53	David Bernholdt, ORNL
10:00-10:30am	*Break*		
10:30-11:30am	Language Interoperability with Babel	85	Gary Kumfert, LLNL
11:30am-12:00pm	A Simple CCA Example	121	Boyana Norris, ANL & Rob Armstrong, SNL
12:00-1:30pm	*Lunch*		
1:30-2:15pm	Writing CCA Components	140	Boyana Norris, ANL & Rob Armstrong, SNL
2:15-3:00pm	Using Ccaffeine	178	Boyana Norris, ANL & Rob Armstrong, SNL
3:00-3:30pm	*Break*		
3:30-4:30pm	A Look at More Complex CCA Applications	203	Lori Freitag Diachin, LLNL
4:30-5:00pm	CCA Status and Plans	246	Lori Freitag Diachin, LLNL

CCA Common Component Architecture

Who Are We?

(And Where Did We Come From?)

- ACTS Toolkit Interoperability Effort (Late 1990's)
  - Part of DOE 2000, Many Tool Integration Projects
    - "One-to-One", Leading to N^2 Solutions… ☹
- Common Component Architecture Forum (1998)
  - Goal: To Develop a General Interoperability Solution
  - Grass Roots Effort to Explore High-Performance Components for Scientific Software
- SciDAC Center for Component Technology for Terascale Simulation Software (CCTTSS, 2001)
  - Part of New DOE Scientific Simulation Program
  - Technology Development in Several Thrust Areas…

CCA Common Component Architecture

The Common Component Architecture (CCA) Forum

- Define Specifications for ***High-Performance*** Scientific Components & Frameworks
- Promote and Facilitate Development of Domain-Specific ***"Standard" Interfaces***
- Goal: ***Interoperability*** between components developed by different expert teams across different institutions
- Quarterly Meetings, Open membership…

Mailing List: *cca-forum@cca-forum.org*

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

CCA Common Component Architecture

Center for Component Technology for Terascale Simulation Software (CCTTSS)

- DOE SciDAC ISIC ($16M over 5 years)
  - SciDAC = Scientific Discovery through Advanced Computing
  - ISIC = Integrated Software Infrastructure Center
  - Funded by Office of Mathematical, Information and Computational Sciences (MICS)
- Develop CCA technology from current prototype stage to full production environment
- Increase understanding of how to use component architectures effectively in HPC environments

- Subset of CCA Forum
- Participants: ANL, LLNL, LANL, ORNL, PNNL, SNL, Indiana University, University of Utah
- Lead PI: Rob Armstrong, SNL rob@sandia.gov
- http://www.cca-forum.org/ccttss/

CCA Common Component Architecture

CCTTSS Research Thrust Areas and Main Working Groups

- Scientific Components
  - Scientific Data Objects

  Lois Curfman McInnes, ANL (curfman@mcs.anl.gov)
- "MxN" Parallel Data Redistribution

  Jim Kohl, ORNL (kohlja@ornl.gov)
- Frameworks
  - Language Interoperability / Babel / SIDL
  - Component Deployment / Repository

  Gary Kumfert, LLNL (kumfert@llnl.gov)
- User Outreach

  David Bernholdt, ORNL (bernholdtde@ornl.gov)

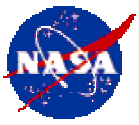
Acknowledgements

- **CCA Forum Tutorial WG**
  - Rob Armstrong, David Bernholdt, Wael Elwasif, Lori Freitag, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, Sameer Shende, Torsten Wilde, Shujia Zhou
  - ANL, JPL, LANL, LLNL, NASA/Goddard, ORNL, SNL, U Oregon

- **And many more contributing to CCA itself…**
  - ANL – Lori Freitag Diachin, Kate Keahey, Jay Larson, Ray Loy, Lois Curfman McInnes, Boyana Norris, …
  - Indiana University - Randall Bramley, Dennis Gannon, …
  - JPL – Dan Katz, …
  - LANL - Craig Rasmussen, Matt Sotille, …
  - LLNL – Lori Freitag Diachin, Tom Epperly, Scott Kohn, Gary Kumfert, …
  - NASA/Goddard – Shujia Zhou
  - ORNL - David Bernholdt, Wael Elwasif, Jim Kohl, Torsten Wilde, …
  - PNNL - Jarek Nieplocha, Theresa Windus, …
  - SNL - Rob Armstrong, Ben Allan, Lori Freitag Diachin, Curt Janssen, Jaideep Ray, …
  - University of Oregon – Allen Malony, Sameer Shende, …
  - University of Utah - Steve Parker, …
  - And others as well …

CCA Common Component Architecture

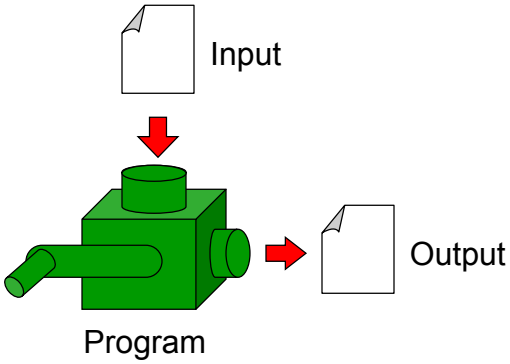
A Pictorial Introduction to Components in Scientific Computing



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

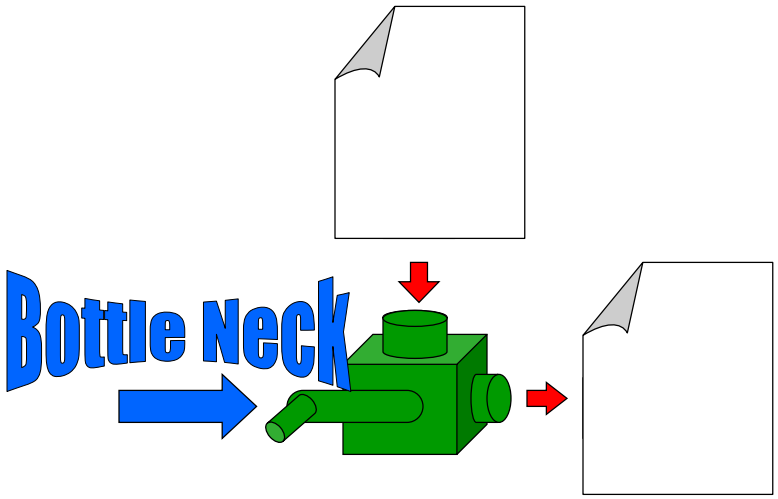
CCA
Common Component Architecture

Once upon a time...



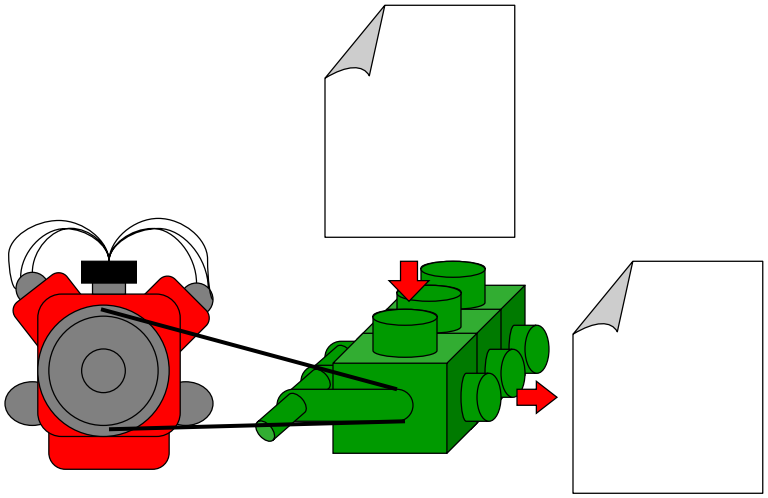
CCA
Common Component Architecture

As Scientific Computing grew...



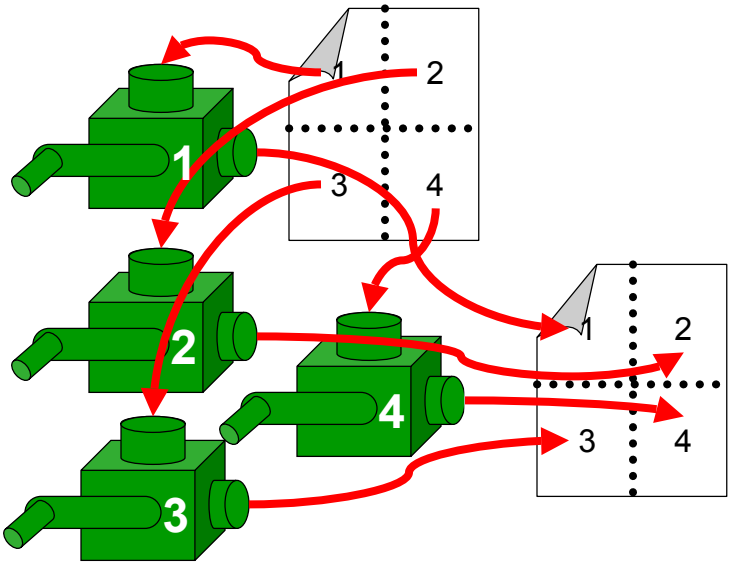
CCA
Common Component Architecture

Tried to ease the bottle neck



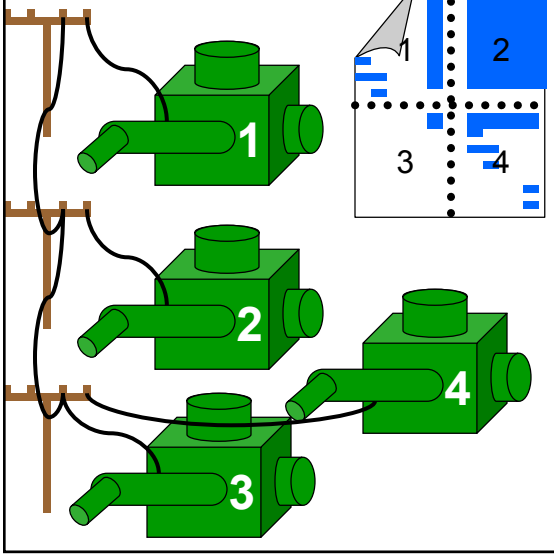
CCA
Common Component Architecture

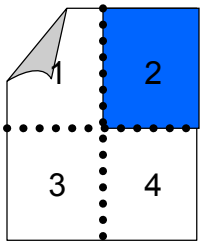
SPMD was born.



CCA
Common Component Architecture

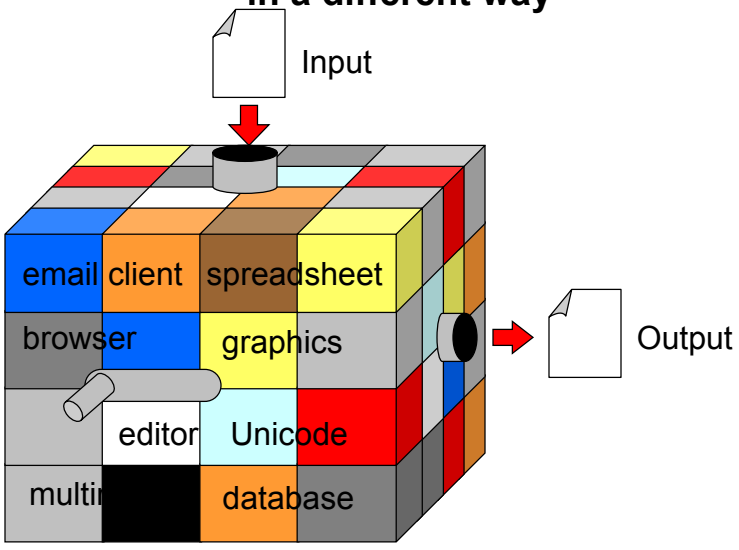
SPMD worked.





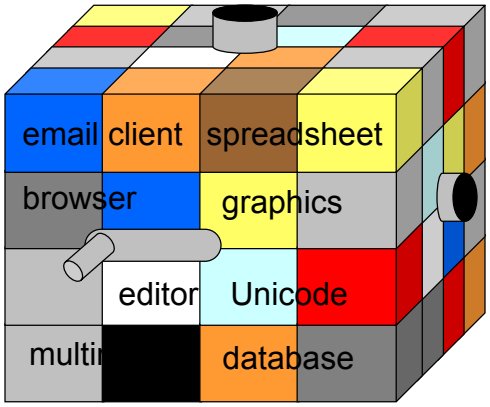
CCA
Common Component Architecture

Meanwhile, corporate computing was growing in a different way



CCA
Common Component Architecture

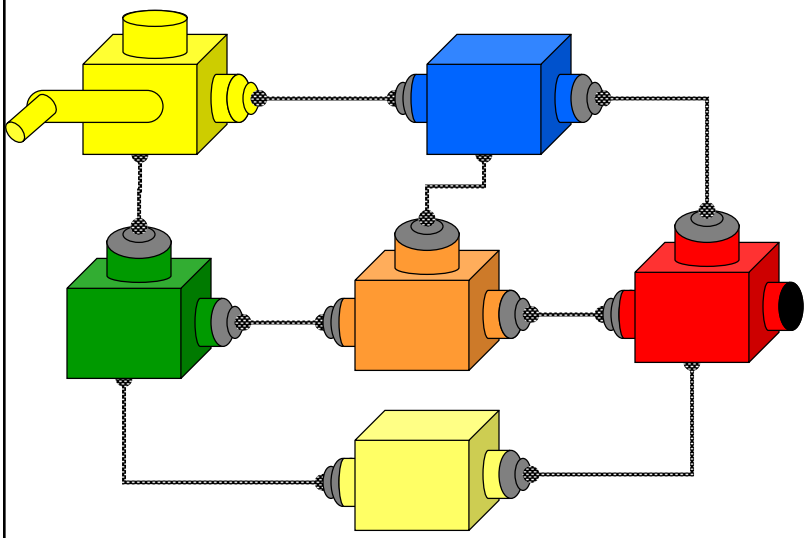
This created a whole new set of problems → complexity



- Interoperability across multiple languages
- Interoperability across multiple platforms
- Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)

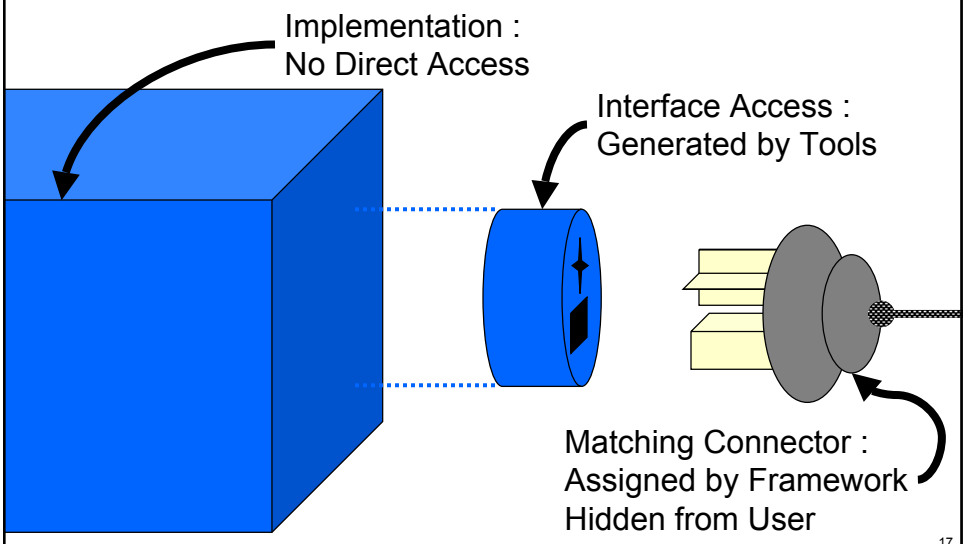
CCA
Common Component Architecture

Component Technology addresses these problems



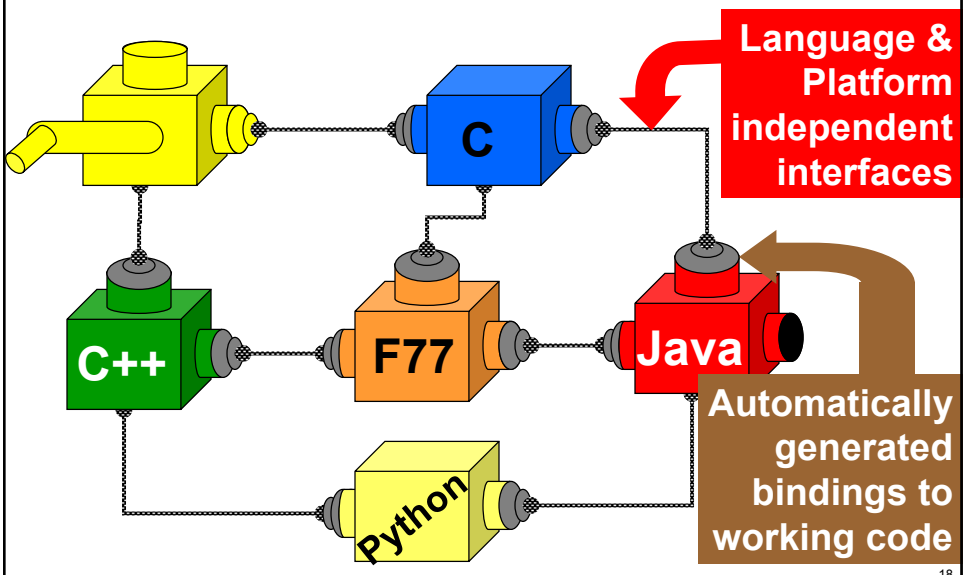
CCA
Common Component Architecture

So what's a component ???



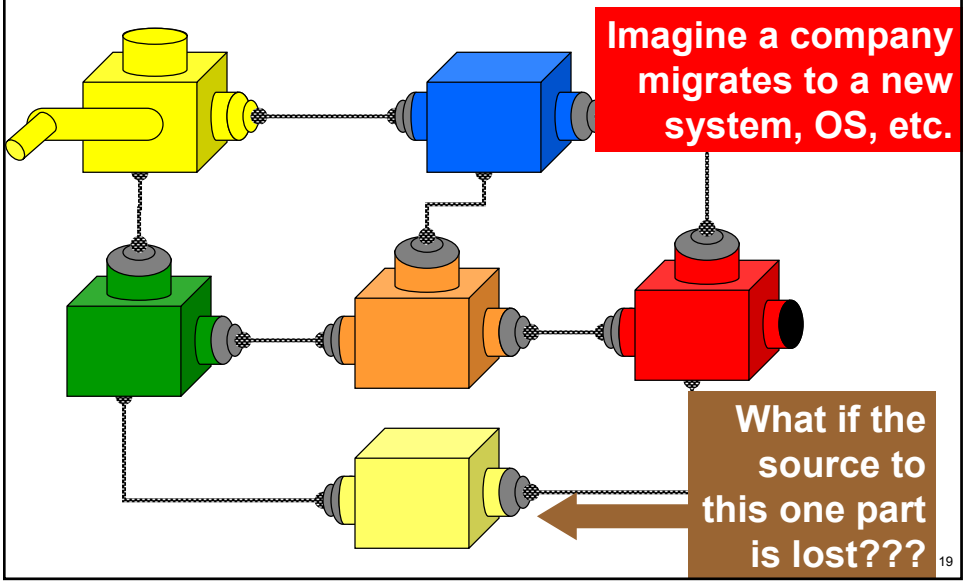
CCA
Common Component Architecture

1. Interoperability across multiple languages



CCA
Common Component Architecture

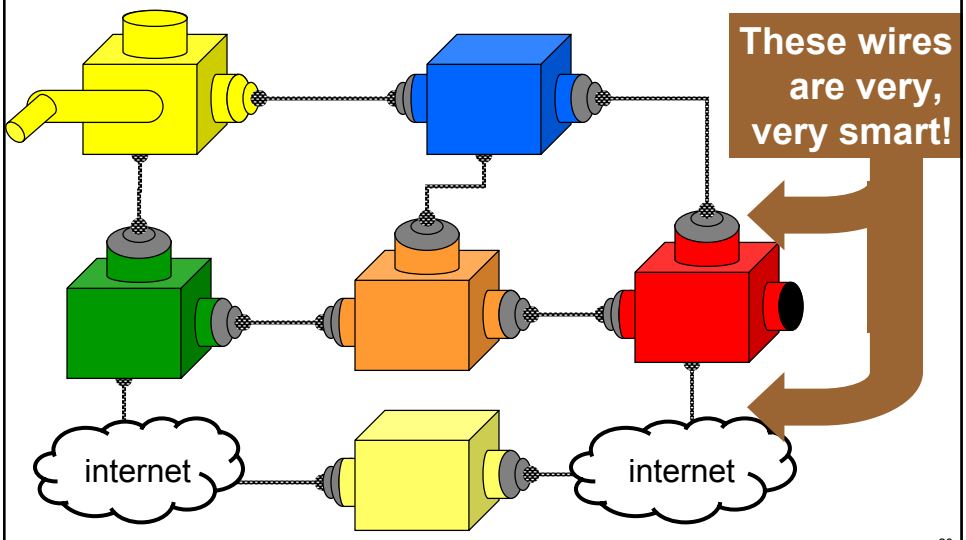
2. Interoperability Across Multiple Platforms



19

CCA
Common Component Architecture

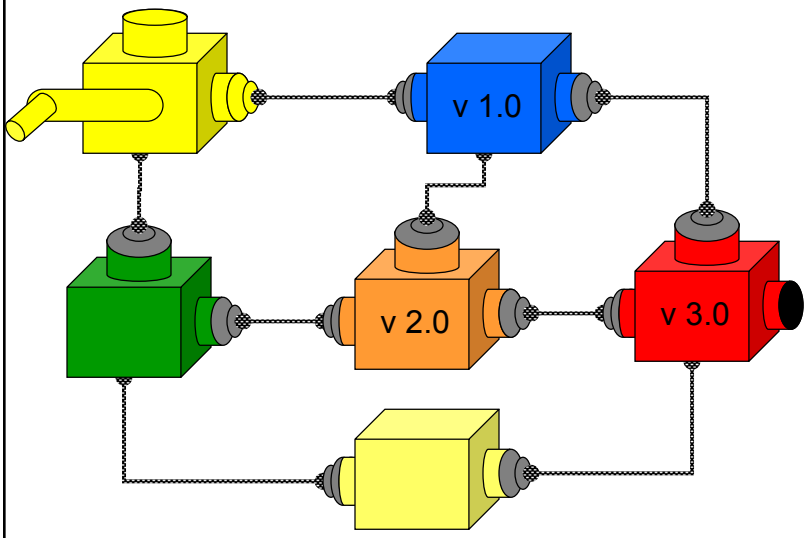
Transparent Distributed Computing



20

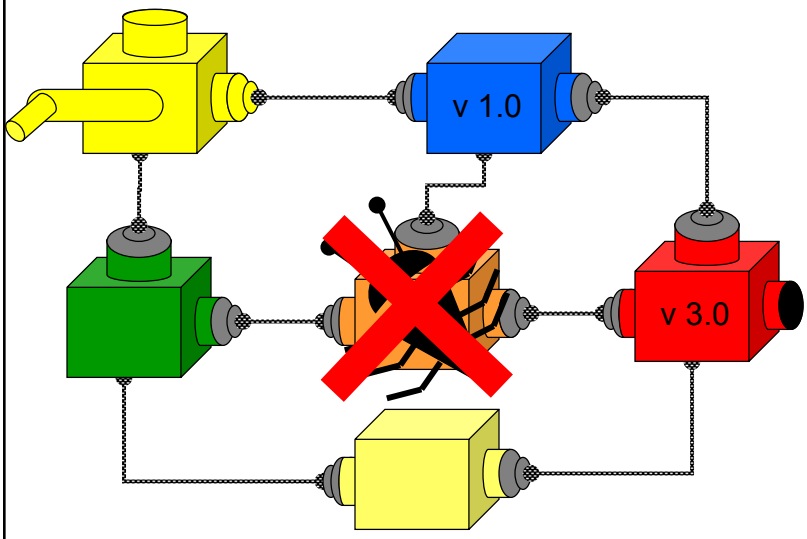
CCA
Common Component Architecture

3. Incremental Evolution With Multiple 3rd party software



CCA
Common Component Architecture

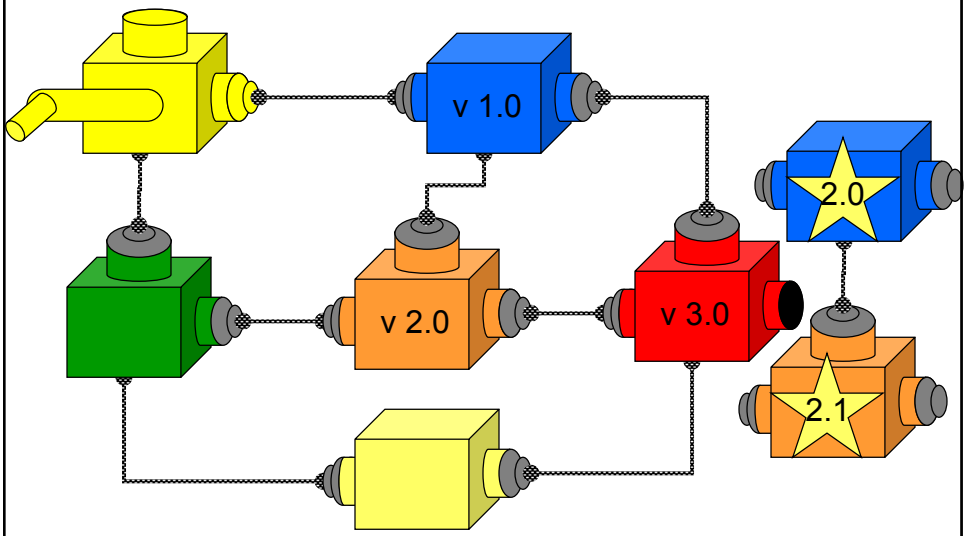
Now suppose you find this bug...



CCA
Common Component Architecture

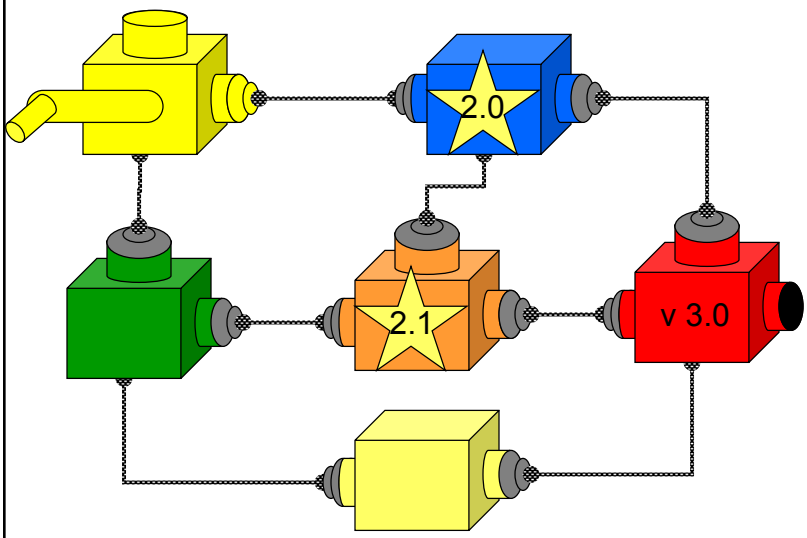
Good news: an upgrade available

Bad news: there's a dependency



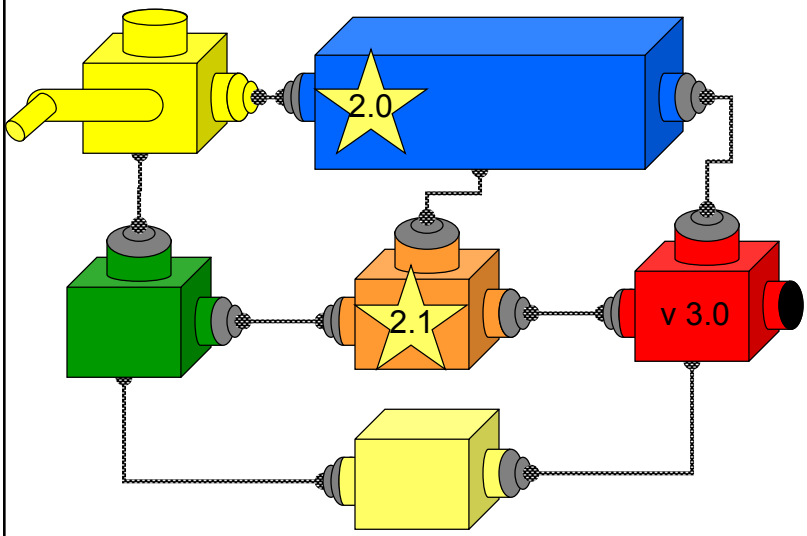
CCA
Common Component Architecture

Great News:
Solvable with Components



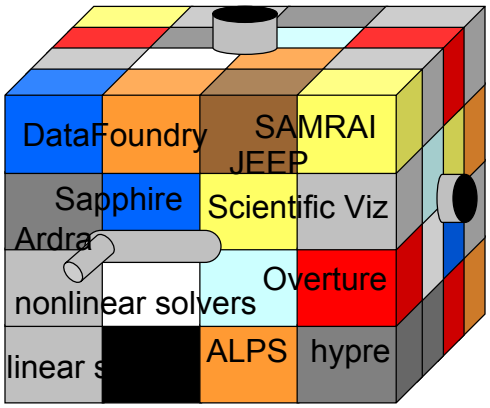
CCA Common Component Architecture

Great News: Solvable with Components



CCA Common Component Architecture

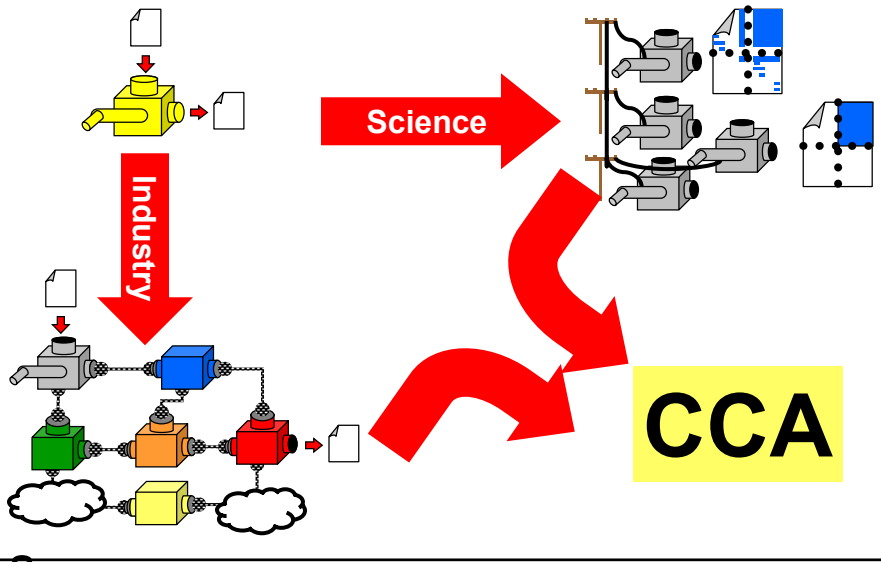
Why Components for Scientific Computing → Complexity



- Interoperability across multiple languages
- Interoperability across multiple platforms
- Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)

CCA
Common Component Architecture

The Model for Scientific Component Programming



CCA
Common Component Architecture

Components for Scientific Computing: An Introduction

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

CCA Common Component Architecture

Goals of This Module

- Introduce basic concepts and vocabulary of component-based software engineering
- Highlight the special demands of high-performance scientific computing on component environments
- Provide a unifying context for the remaining talks
  - And to consider what components might do for your applications

CCA Common Component Architecture

Motivation: Modern Scientific Software Engineering Challenges

- **Productivity**
  - Time to first solution (prototyping)
  - Time to solution ("production")
  - Software infrastructure requirements ("other stuff needed")
- **Complexity**
  - Increasingly sophisticated models
  - Model coupling – multi-scale, multi-physics, etc.
  - "Interdisciplinarity"
- **Performance**
  - Increasingly complex algorithms
  - Increasingly complex computers
  - Increasingly demanding applications

CCA Common Component Architecture

Motivation: For Library Developers

- People want to use your software, but need wrappers in languages you don't support
  - Many component models provide language interoperability
- Discussions about standardizing interfaces are often sidetracked into implementation issues
  - Components separate interfaces from implementation
- You want users to stick to your published interface and prevent them from stumbling (prying) into the implementation details
  - Most component models actively enforce the separation

CCA Common Component Architecture

Motivation: For Application Developers and Users

- You have difficulty managing multiple third-party libraries in your code
- You (want to) use more than two languages in your application
- Your code is long-lived and different pieces evolve at different rates
- You want to be able to swap competing implementations of the same idea and test without modifying any of your code
- You want to compose your application with some other(s) that weren't originally designed to be combined

Some Observations About Software…

- "The complexity of software is an essential property, not an accidental one." *[Brooks]*
  - We can't get rid of complexity
- "Our failure to master the complexity of software results in projects that are late, over budget, and deficient in their stated requirements." *[Booch]*
  - We must find ways to manage it

More Observations…

- "A complex system that works is invariably found to have evolved from a simple system that worked… A complex system designed from scratch never works and cannot be patched up to make it work." *[Gall]*
  - Build up from simpler pieces
- "The best software is code you don't have to write" *[Jobs]*
  - Reuse code wherever possible

CCA Common Component Architecture

Not All Complexity is "Essential"

- An example of how typical development practices can exacerbate the complexity of software development…
- At least 41 different Fast Fourier Transform (FFT) libraries:
  - see, http://www.fftw.org/benchfft/doc/ffts.html
- Many (if not all) have different interfaces
  - different procedure names and different input and output parameters
- Example: SUBROUTINE FOUR1(DATA, NN, ISIGN)
  - "Replaces DATA by its discrete Fourier transform (if ISIGN is input as 1) or replaces DATA by NN times its inverse discrete Fourier transform (if ISIGN is input as -1). DATA is a complex array of length NN or, equivalently, a real array of length 2*NN. NN MUST be an integer power of 2 (this is not checked for!)."

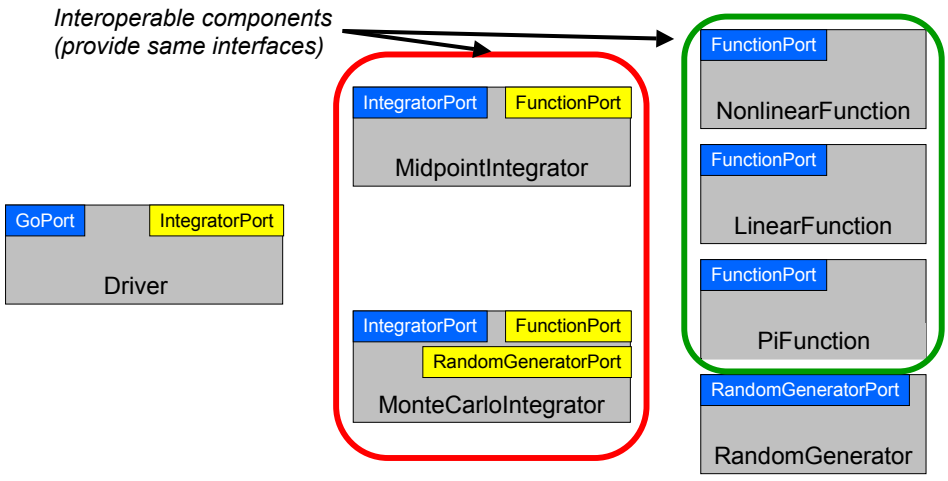
CCA Common Component Architecture

Component-Based Software Engineering

- CBSE methodology is emerging, especially from business and internet areas
- **Software productivity**
  - Provides a "plug and play" application development environment
  - Many components available "off the shelf"
  - Abstract interfaces facilitate reuse and interoperability of software
- **Software complexity**
  - Components encapsulate much complexity into "black boxes"
  - Plug and play approach simplifies applications
  - Model coupling is natural in component-based approach
- **Software performance** (indirect)
  - Plug and play approach and rich "off the shelf" component library simplify changes to accommodate different platforms

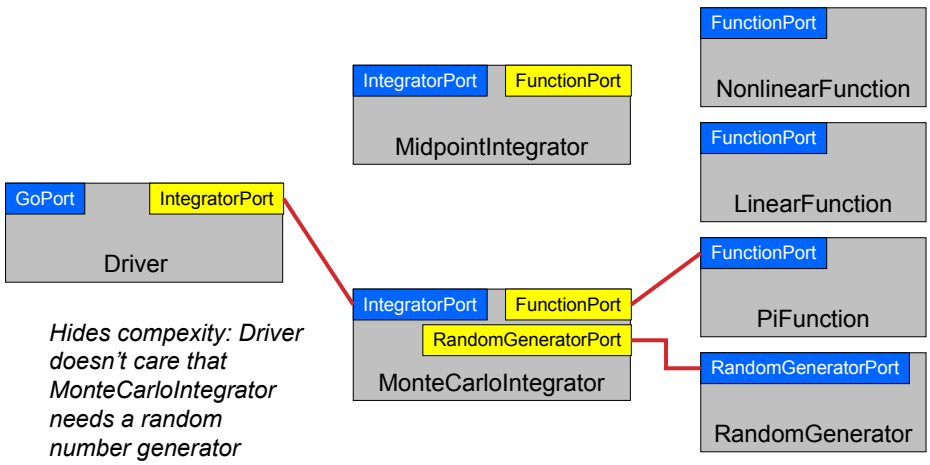
CCA Common Component Architecture

A Simple Example: Numerical Integration Components



CCA Common Component Architecture

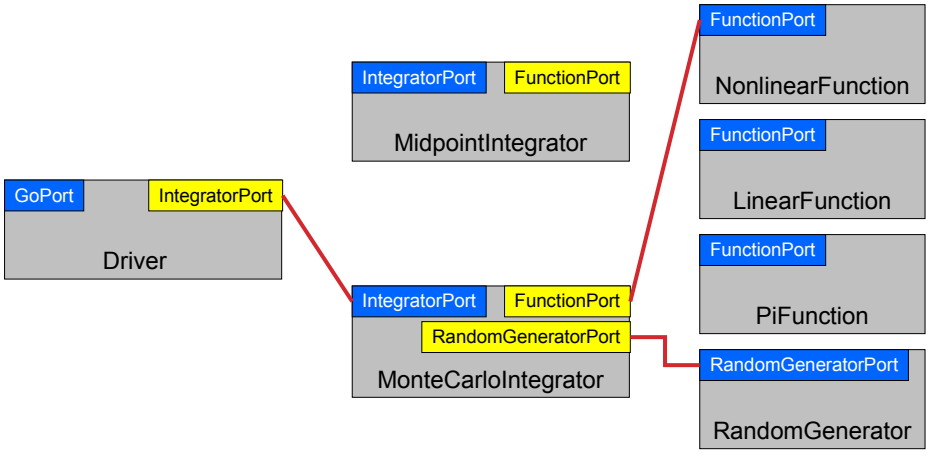
An Application Built from the Provided Components



Hides compexity: Driver doesn't care that MonteCarloIntegrator needs a random number generator

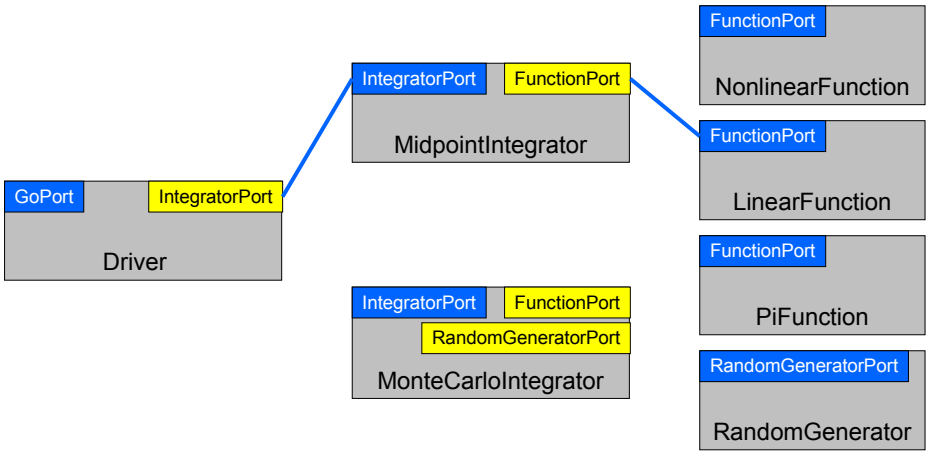
CCA Common Component Architecture

Another Application…



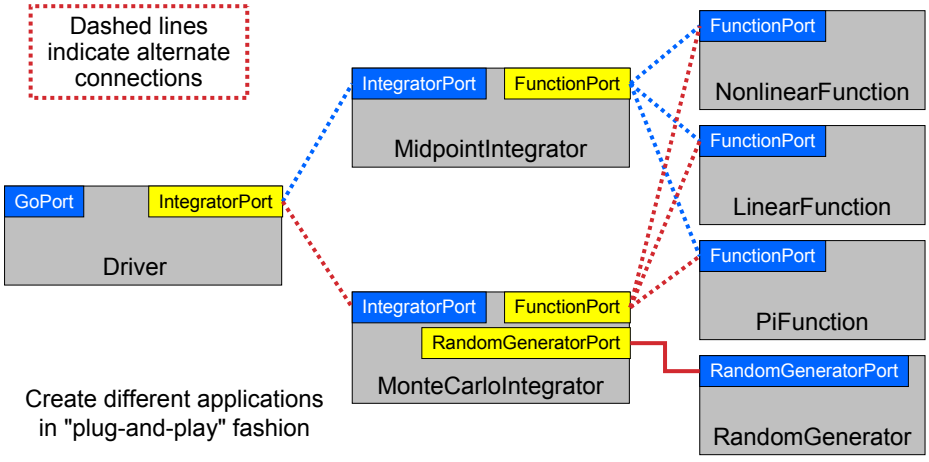
CCA Common Component Architecture

Application 3…



CCA Common Component Architecture

And Many More…



CCA Common Component Architecture

What *are* Components?

- No universally accepted definition…yet
- **A unit of software development/deployment/reuse**
  - i.e. has interesting functionality
  - Ideally, functionality someone else might be able to (re)use
  - Can be developed independently of other components
- **Interacts with the outside world *only* through well-defined interfaces**
  - Implementation is opaque to the outside world
  - Components *may* maintain state information
  - But external access to state info must be through an interface (*not a common block*)
  - File-based interactions can be recast using an “I/O component”
- **Can be composed with other components**
  - “Plug and play” model to build applications
  - Composition based on interfaces

CCA Common Component Architecture

What is a Component Architecture?

- A set of standards that allows:
  - Multiple groups to write units of software (components)…
  - And have confidence that their components will work with other components written in the same architecture
- These standards define…
  - The rights and responsibilities of a component
  - How components express their interfaces
  - The environment in which are composed to form an application and executed (framework)
  - The rights and responsibilities of the framework

CCA Common Component Architecture

Interfaces, Interoperability, and Reuse

- Interfaces define how components interact…
- Therefore interfaces are key to interoperability and reuse of components
- In many cases, “any old interface” will do, but…
- General plug and play interoperability requires multiple implementations providing the same interface
- Reuse of components occurs when they provide interfaces (functionality) needed in multiple applications

CCA Common Component Architecture

Designing for Reuse, Implications

- Designing for interoperability and reuse requires "standard" interfaces
  - Typically domain-specific
  - "Standard" need not imply a formal process, may mean "widely used"
- Generally means collaborating with others
- *Higher* initial development cost (amortized over multiple uses)
- Reuse implies longer-lived code
  - thoroughly tested
  - highly optimized
  - improved support for multiple platforms

CCA Common Component Architecture

Relationships: Components, Objects, and Libraries

- Components are typically discussed as objects or collections of objects
  - Interfaces generally designed in OO terms, but…
  - Component internals need not be OO
  - OO languages are not required
- Component environments can enforce the use of published interfaces (prevent access to internals)
  - Libraries can not
- It is possible to load several instances (versions) of a component in a single application
  - Impossible with libraries
- Components *must* include some code to interface with the framework/component environment
  - Libraries and objects do not

CCA Common Component Architecture

Domain-Specific Frameworks vs Generic Component Architectures

Domain-Specific

- Often known as "frameworks"
- Provide a significant software infrastructure to support applications in a given domain
  - Often attempts to generalize an existing large application
- Often hard to adapt to use outside the original domain
  - Tend to assume a particular structure/workflow for application
- Relatively common

Generic

- Provide the infrastructure to hook components together
  - Domain-specific infrastructure can be built as components
- Usable in many domains
  - Few assumptions about application
  - More opportunities for reuse
- Better supports model coupling across traditional domain boundaries
- Relatively rare at present
  - Commodity component models often not so useful in HPC scientific context

CCA Common Component Architecture

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 Common Component Architecture

Commodity Component Models

- CORBA, COM, Enterprise JavaBeans
  - Arise from business/internet software world
- Componentization requirements can be high
- Can impose significant performance overheads
- No recognition of tightly-coupled parallelism
- May be platform specific
- May have language constraints
- May not support common scientific data types

CCA Common Component Architecture

The "Sociology" of Components

- Components need to be shared to be truly useful
  - Sharing can be at several levels
    - Source, binaries, remote service
  - Various models possible for intellectual property/licensing
    - Components with different IP constraints can be mixed in a single application
- Peer component models facilitate collaboration of groups on software development
  - Group decides overall architecture and interfaces
  - Individuals/sub-groups create individual components

CCA Common Component Architecture

Who Writes Components?

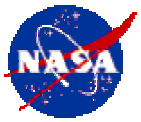
- "Everyone" involved in creating an application can/should create components
  - Domain scientists as well as computer scientists and applied mathematicians
  - Most will also use components written by other groups
- Allows developers to focus on their interest/specialty
  - Get other capabilities via reuse of other's components
- Sharing components within scientific domain allows everyone to be more productive
  - Reuse instead of reinvention
- As a unit of publication, a well-written and –tested component is like a high-quality library
  - Should receive same degree of recognition
  - Often a more appropriate unit of publication/recognition than an entire application code

CCA Common Component Architecture

Summary

- Components are a software engineering tool to help address software productivity and complexity
- Important concepts: components, interfaces, frameworks, composability, reuse
- Scientific component environments come in "domain specific" and "generic" flavors
- Scientific HPC imposes special demands on component environments
  - Which commodity tools may have trouble with

Common Component Architecture Concepts



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

Goals

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

CCA Common Component Architecture

What is the CCA? (User View)

- A component model specifically designed for high-performance scientific computing
- Minimalist approach makes it easier to componentize existing software
- A *tool* to enhance the productivity of scientific programmers
  - Make the hard things easier, make some intractable things tractable
  - Support & promote reuse & interoperability
  - Not a magic bullet

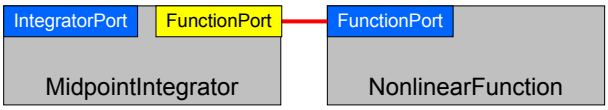
CCA Common Component Architecture

What is the CCA? (Technical View)

- CCA is a *specification* of a component environment
  - A design pattern
  - Defines rights and responsibilities of a CCA component
  - Defines how CCA components express their interfaces
  - Defines rights and responsibilities of a CCA framework
- "CCA compliant" means conforming to the specification
  - Doesn't require using any of our code
- CCA specification is decided by the CCA Forum
  - Membership in the CCA Forum is open to all

CCA Common Component Architecture

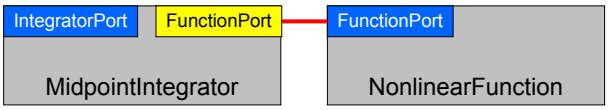
CCA Concepts: Components



- Components provide/use one or more ports
  - A component with no ports isn't very interesting
- Components include some code which interacts with a CCA framework

CCA Common Component Architecture

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 ("Provides" Port)
- Components may *use* ports – call methods or subroutines in the port ("Uses" Port)
- Links between ports denote a procedural (caller/callee) relationship, ***not dataflow!***
  - e.g., FunctionPort could contain: *evaluate(in Arg, out Result)*

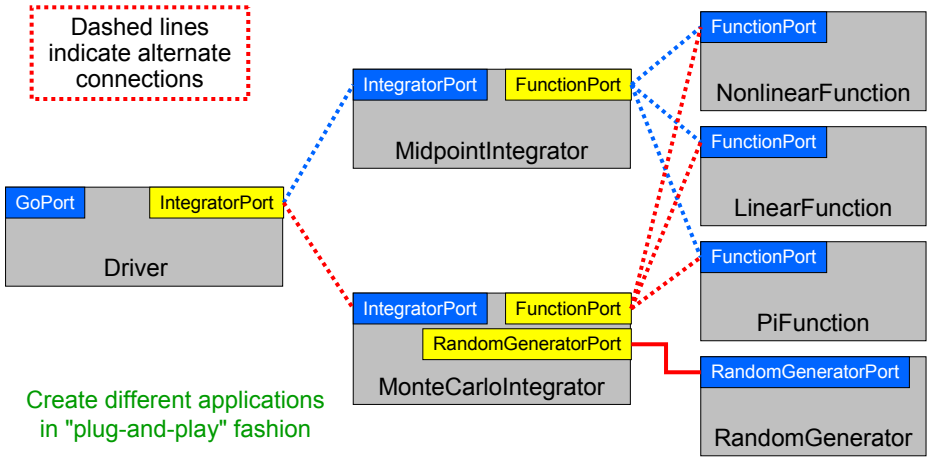
CCA Common Component Architecture

CCA Concepts: Frameworks

- The framework provides the means to "hold" components and compose them into applications
- Frameworks allow connection of ports without exposing component implementation details
- Frameworks provide a small set of standard services to components
- *Currently:* specific frameworks support specific computing models (parallel, distributed, etc.)
- *Future:* full flexibility through integration or interoperation

CCA Common Component Architecture

Components and Ports in the Integrator Example



CCA Common Component Architecture

Writing Components

- Components…
  - Inherit from **gov.cca.Component**
    - Implement **setServices** method to register ports this component will provide and use
  - Implement the ports they provide
  - Use ports on other components
    - **getPort/releasePort** from framework **Services** object
- Interfaces (ports) extend **gov.cca.Port**

***Much* more detail later in the tutorial!**

CCA Common Component Architecture

Adapting Existing Code into Components

Suitably structured code (programs, libraries) should be relatively easy to adapt to the CCA. Here's how:

1. Decide level of componentization
   - Can evolve with time (start with coarse components, later refine into smaller ones)
2. Define interfaces and write wrappers between them and existing code
3. Add framework interaction code for each component
   - **setServices**
4. Modify component internals to use other components as appropriate
   - **getPort**, **releasePort** and method invocations

CCA Common Component Architecture

Writing Frameworks

- ***There is no reason for most people to write frameworks – just use the existing ones!***
- Frameworks must provide certain ports…
  - **ConnectionEventService**
    - Informs the component of connections
  - **AbstractFramework**
    - Allows the component to *behave as a framework*
  - **BuilderService**
    - Instantiate components & connect ports
  - **ComponentRepository**
    - A default place where components are found
- Frameworks must be able to load components
  - Typically shared object libraries, can be statically linked
- Frameworks must provide a way to compose applications from components

CCA Common Component Architecture

Component Lifecycle

We'll look at actual code in next tutorial module

- **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

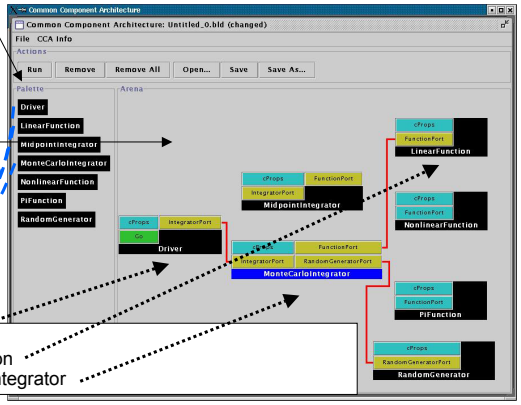
In an application, individual components may be in different phases at different times

Steps may be under human or software control

User Viewpoint: Loading and Instantiating Components

- Components are code + 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

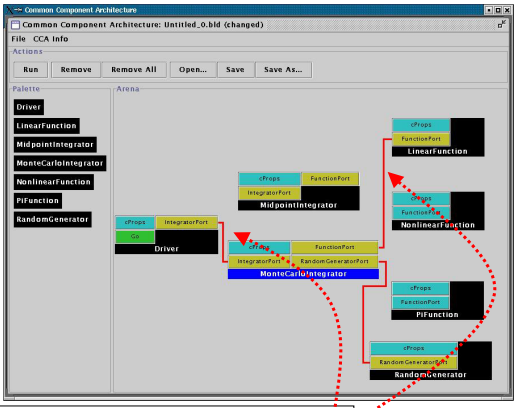
- Details are framework-specific!
- Ccaffeine currently provides both command line and GUI approaches



create	Driver	Driver
create	LinearFunction	LinearFunction
create	MonteCarloIntegrator	MonteCarloIntegrator

User Connects Ports

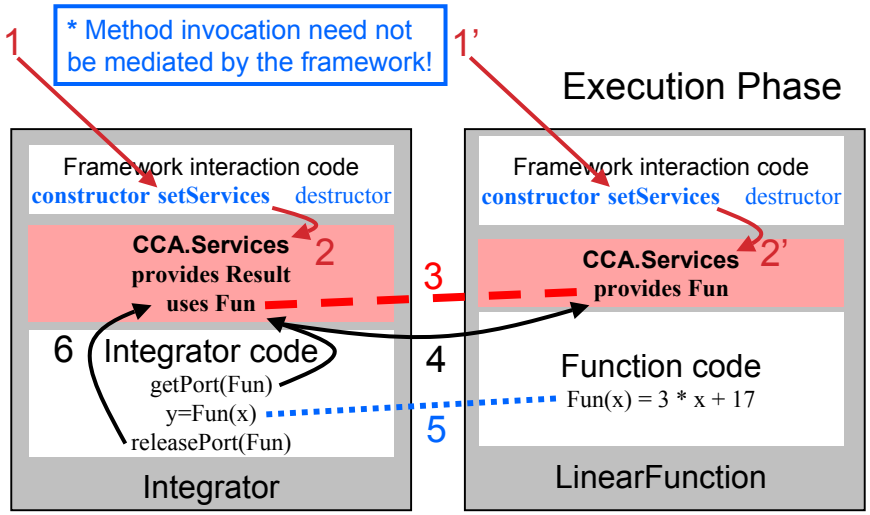
- Can only connect uses & provides
  - Not uses/uses or provides/provides
- Ports connected by type, not name
  - Port names must be unique within component
  - Types must match across components
- Framework puts info about *provider* of port into *using component's* Services object



connect	Driver	IntegratorPort	MonteCarloIntegrator	IntegratorPort
connect	MonteCarloIntegrator	FunctionPort	LinearFunction	FunctionPort
…				

CCA Common Component Architecture

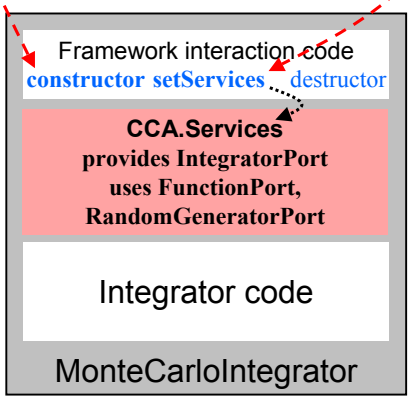
Framework Mediates *Most** Component Interactions



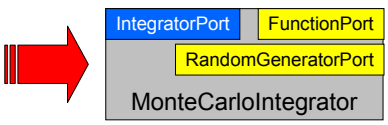
CCA Common Component Architecture

Component's View of Instantiation

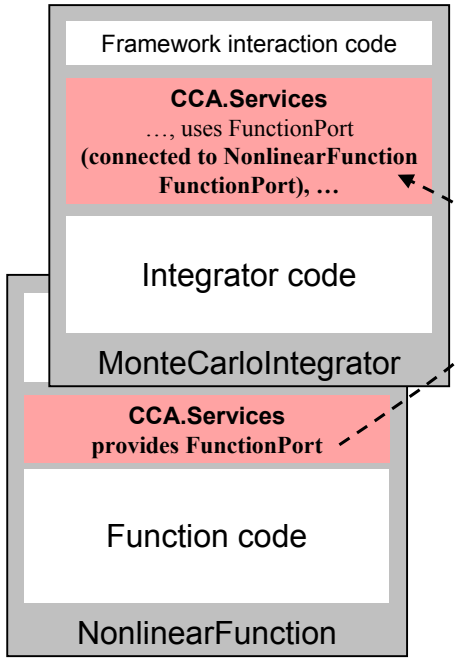
- Framework calls component's constructor
- Component initializes internal data, etc.
  - Knows *nothing* outside itself



- 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



CCA Common Component Architecture



Component's View of Connection

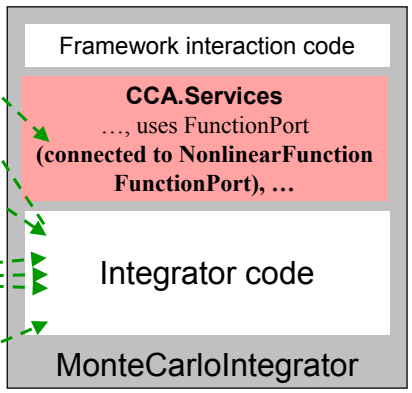
- Framework puts info about provider into user component's Services object
  - ***MonteCarloIntegrator***'s Services object is aware of connection
  - ***NonlinearFunction*** is not!
- ***MCI***'s integrator code cannot yet call functions on FunctionPort

CCA Common Component Architecture

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.

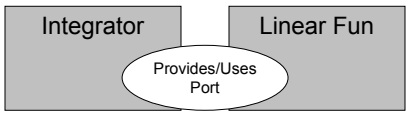
  sum = sum + function->evaluate(x)
- Release port



CCA Common Component Architecture

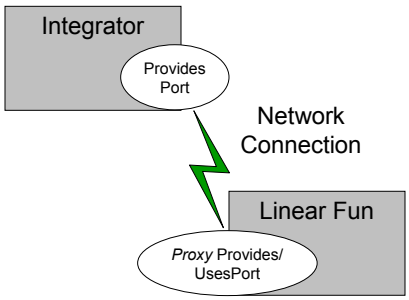
CCA Supports Local, Parallel and Distributed Computing

- "Direct connection" preserves high performance of local ("in-process") components
  - Framework makes *connection*
  - But is not involved in *invocation*



Direct Connection

- Distributed computing has same uses/provides pattern, but framework intervenes between user and provider
  - Framework provides a *proxy* provides port local to the *uses* port
  - Framework conveys invocation from proxy to actual provides port



CCA Common Component Architecture

CCA Concepts: "Direct Connection" Maintains Local Performance

- Calls *between* components equivalent to a C++ virtual function call: lookup function location, invoke it
  - Cost equivalent of ~2.8 F77 or C function calls
  - ~48 ns vs 17 ns on 500 MHz Pentium III Linux box
- Language interoperability can impose additional overheads
  - Some arguments require conversion
  - Costs vary, but small for typical scientific computing needs
- Calls within components have no CCA-imposed overhead
- **Implications**
  - Be aware of costs
  - Design so inter-component calls do enough work that overhead is negligible

CCA Common Component Architecture

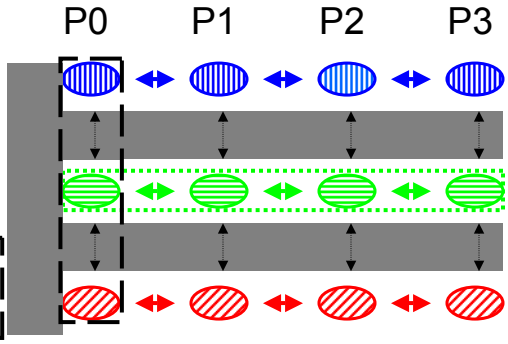
How Does Direct Connection Work?

- 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
- All this happens "automatically" – user just sees high performance
  - *Description reflects Ccaffeine implementation, but similar or identical mechanisms are in other direct connect fwks*
- *Many CORBA implementations offer a similar approach to improve performance, but using it violates the CORBA standards!*

CCA Common Component Architecture

CCA Concepts: Framework Stays "Out of the Way" of Component Parallelism

- 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

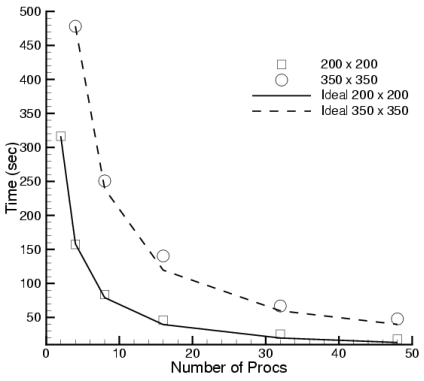
Framework: Gray

MCMD/MPMD also supported

Other component models ignore parallelism entirely

Scalability of Scientific Data Components in CFRFS Combustion Applications

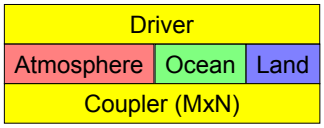
- Investigators: S. Lefantzi, J. Ray, and H. Najm (SNL)
- Uses GrACEComponent, CvodesComponent, etc.
- Shock-hydro code with no refinement
- 200 x 200 & 350 x 350 meshes
- Cplant cluster
  - 400 MHz EV5 Alphas
  - 1 Gb/s Myrinet
- Negligible component overhead
- Worst perf : 73% scaling efficiency for 200x200 mesh on 48 procs



Reference: S. Lefantzi, J. Ray, and H. Najm, Using the Common Component Architecture to Design High Performance Scientific Simulation Codes, *Proc of Int. Parallel and Distributed Processing Symposium*, Nice, France, 2003.

"Multiple-Component Multiple-Data" Applications in CCA

- Simulation composed of multiple SCMD sub-tasks
- Usage Scenarios:
  - Model coupling (e.g. Atmosphere/Ocean)
  - General multi-physics applications
  - Software licensing issues



- Approaches
  - Run single parallel framework
    - Driver component that partitions processes and builds rest of application as appropriate (through BuilderService)
  - Run multiple parallel frameworks
    - Link through specialized communications components (e.g. MxN)
    - Link as components (through AbstractFramework service; highly experimental at present)

CCA Common Component Architecture

MCMD Within A Single Framework

Working examples available using Ccaffeine framework, with driver coded in Python

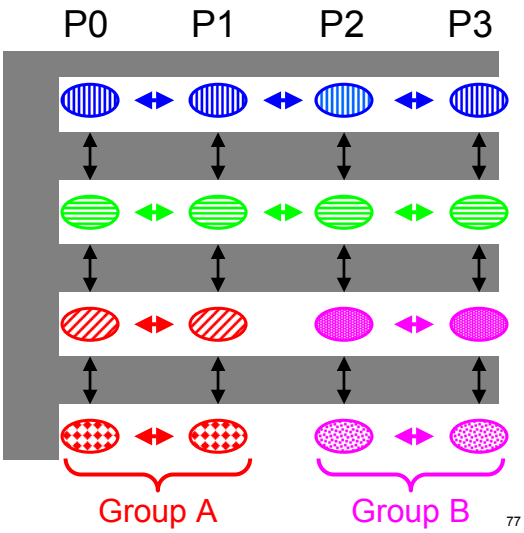
Framework

Application driver & MCMD support component

Components on all processes

Components only on process group A

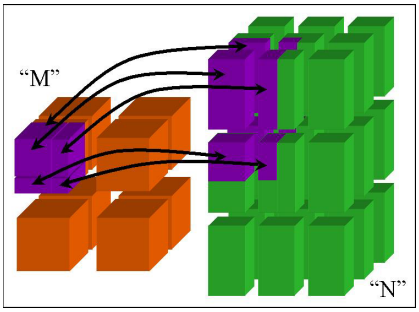
Components only on process group B



CCA Common Component Architecture

CCA Concepts: MxN Parallel Data Redistribution

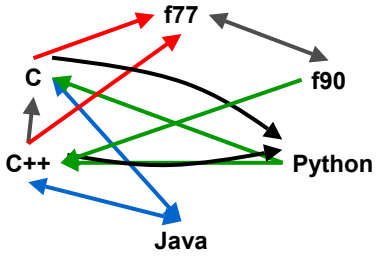
- Share Data Among Coupled Parallel Models
  - Disparate Parallel Topologies (M processes vs. N)
  - e.g. Ocean & Atmosphere, Solver & Optimizer…
  - e.g. Visualization (Mx1, increasingly, MxN)



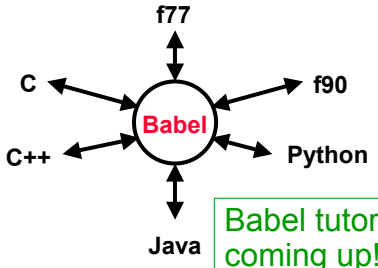
Research area -- tools under development

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!

Few other component models support all languages and data types important for scientific computing

Performance Issues (Redux)

- No CCA overhead on calls within components
- CCA-related overheads on calls to other ports
  - Invocation cost (small for direct connection)
  - Language interoperability costs ("translate" some data types)
  - Design application architecture to minimize overheads
    - Methods in ports should do enough work to amortize overheads
    - Language costs can be minimized for most scientific computing
- No CCA overhead on parallel interactions
- Costs for distributed computing depend on network protocols, etc.

CCA Common Component Architecture

Advanced CCA Concepts

- Components are peers
  - Application architecture determines relationships, not CCA specification
- Frameworks provide a BuilderService which allows programmatic composition of components
- Frameworks may present themselves as components to other frameworks
- A “traditional” application can treat a CCA framework as a library

CCA Common Component Architecture

What the CCA isn’t…

- CCA doesn’t specify who owns “main”
  - CCA components are peers
  - Up to application to define component relationships
    - “Driver component” is a common design pattern
- CCA doesn’t specify a parallel programming environment
  - Choose your favorite
  - Mix multiple tools in a single application
- CCA doesn’t specify I/O
  - But it gives you the infrastructure to create I/O components
  - Use of stdio may be problematic in mixed language env.
- CCA doesn’t specify interfaces
  - But it gives you the infrastructure to define and enforce them
  - CCA Forum supports & promotes “standard” interface efforts
- CCA doesn’t require (but does support) separation of algorithms/physics from data

CCA Common Component Architecture

What the CCA *is…*

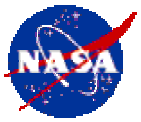
- CCA is a *specification* for a component environment
  - Fundamentally, a design pattern
  - Multiple "reference" implementations exist
  - Being used by applications
- CCA increases productivity
  - Supports and promotes software interopability and reuse
  - Provides "plug-and-play" paradigm for scientific software
- CCA offers the flexibility to architect your application as you think best
  - Doesn't dictate component relationships, programming models, etc.
  - Minimal performance overhead
  - Minimal cost for incorporation of existing software
- CCA provides an environment in which domain-specific application frameworks can be built
  - While retaining opportunities for software reuse at multiple levels

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)

Language Interoperable CCA Components via



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

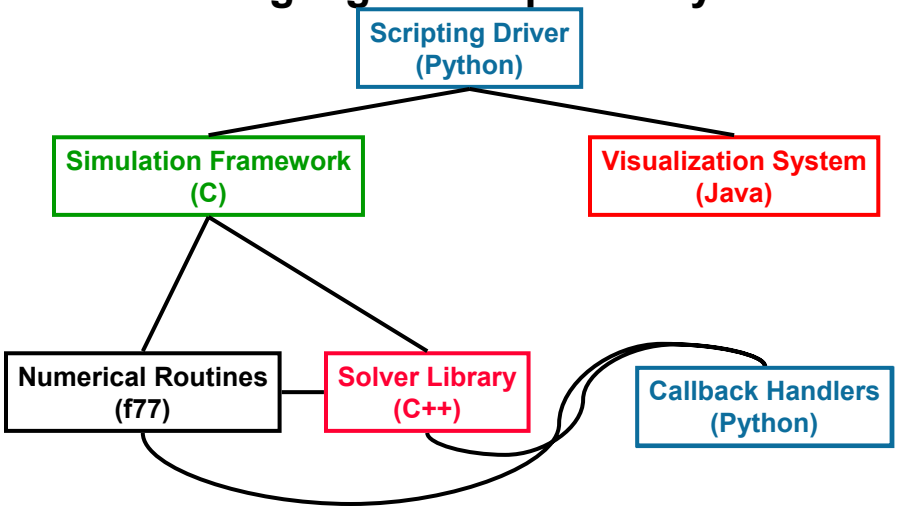
Goal of This Module

Legacy codes → Babelized CCA Components

- Introduction To:
  - Babel
  - SIDL
- See Babel in use
  - "Hello World" example
  - Legacy Code (Babel-wrapped MPI)
  - CCA Tutorial Example (Numerical Integration)
- Relationship between Babel & CCA

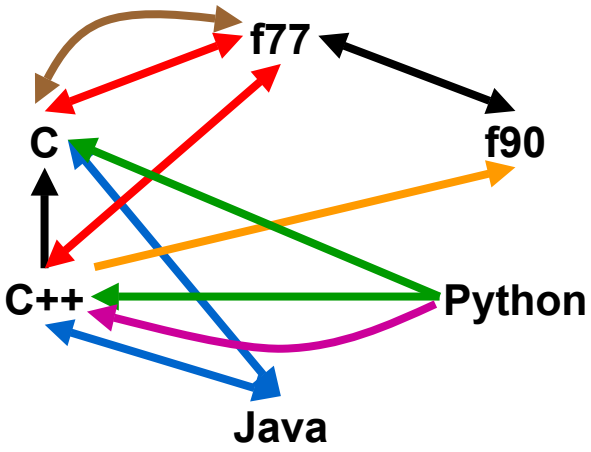
CCA Common Component Architecture

What I mean by "Language Interoperability"



CCA Common Component Architecture

One reason why mixing languages is hard



Native

cfortran.h

SWIG

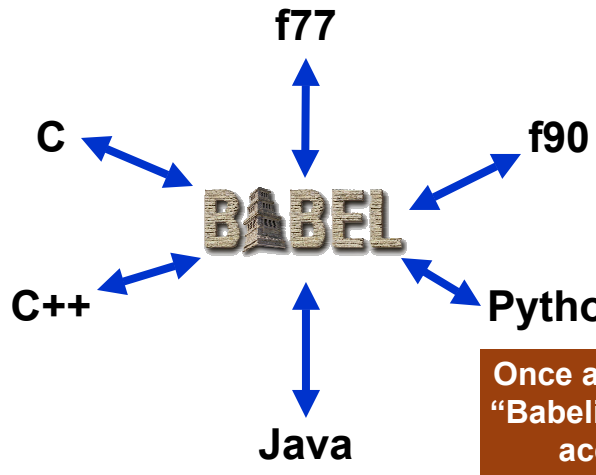
JNI

Siloon

Chasm

Platform Dependent

Babel makes all supported languages peers



This is not a Lowest Common Denominator Solution!

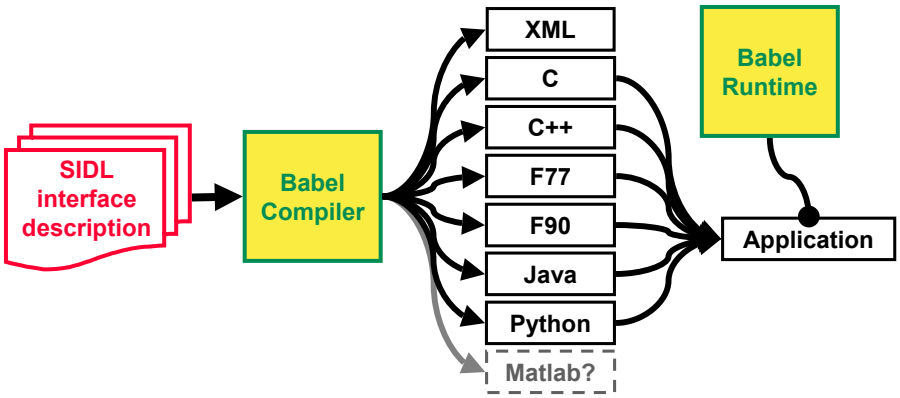
Once a library has been "Babelized" it is equally accessable from all supported languages

Babel Module's Outline

- Introduction
- Babel Basics
  - How to use Babel in a "Hello World" Example
  - SIDL Grammar
  - Example: Babel & Legacy Code
- Babel & CCA
  - Relationship between them
  - How to write a Babelized CCA Component

CCA Common Component Architecture

Babel's Two Parts: Code Generator + Runtime Library



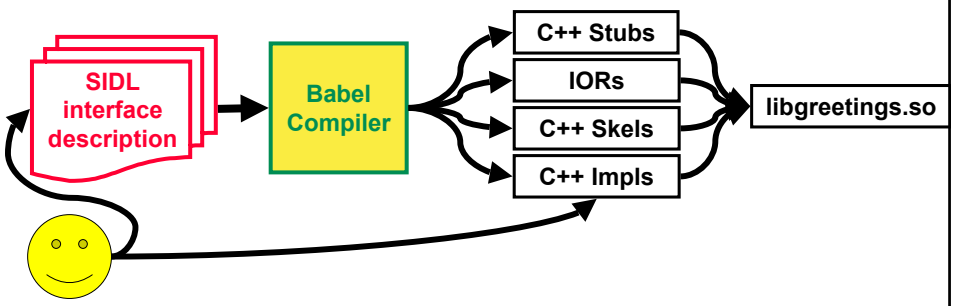
CCA Common Component Architecture

greetings.sidl: A Sample SIDL File

```
package greetings version 1.0 {
    interface Hello {
        void setName( in string name );
        string sayIt ( );
    }
    class English implements-all Hello {  }
}
```

CCA Common Component Architecture

Library Developer Does This...



1. `babel --server=C++ greetings.sidl`
2. Add implementation details
3. Compile & Link into Library/DLL

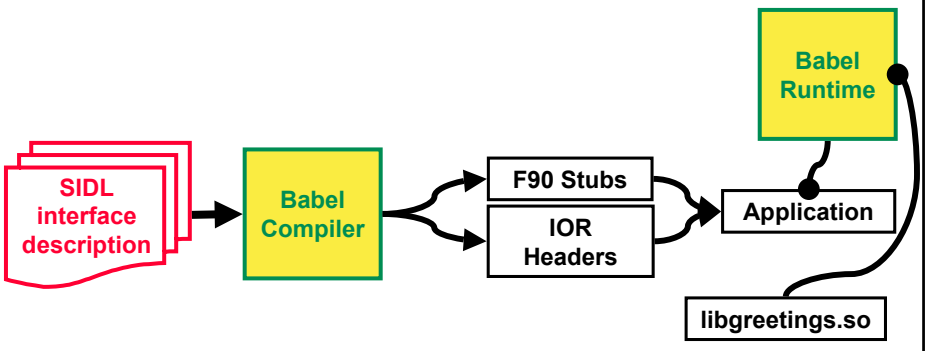
CCA Common Component Architecture

Adding the Implementation

```
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)
}
```

Library User Does This...



1. `babel --client=F90 greetings.sidl`
2. Compile & Link generated Code & Runtime
3. Place DLL in suitable location

F90/Babel "Hello World" Application

```
program helloclient
  use greetings_English
  implicit none
  type(greetings_English_t) :: obj
  character (len=80)        :: msg
  character (len=20)        :: name

  name='World'
  call new( obj )
  call setName( obj, name )
  call sayIt( obj, msg )
  call deleteRef( obj )
  print *, msg

end program helloclient
```

These subroutines come from directly from the SIDL

Some other subroutines are "built in" to every SIDL class/interface

CCA Common Component Architecture

SIDL Grammar (1/3): Packages and Versions

- Packages can be nested

```
package foo version 0.1 { package bar { ... } }
```

- Versioned Packages
  - defined as packages with explicit version number OR packages enclosed by a versioned package
  - Reentrant by default, but can be declared final
  - May contain interfaces, classes, or enums
- Unversioned Packages
  - Can only enclose more packages, not types
  - Must be re-entrant. Cannot be declared final

CCA Common Component Architecture

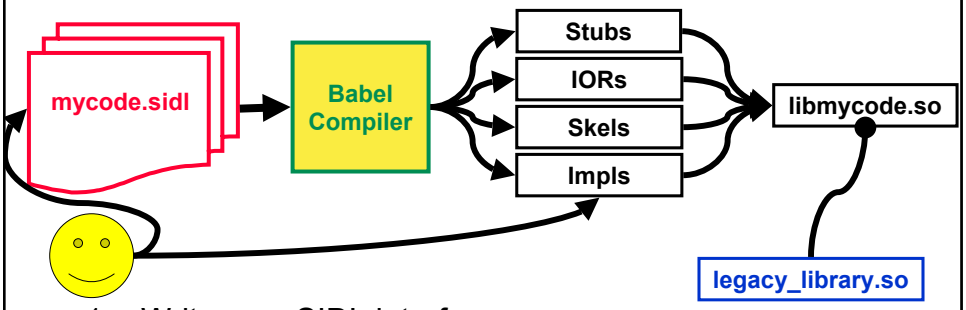
SIDL Grammar (2/3): Classes & Interfaces

- SIDL has 3 user-defined objects
  - Interfaces – APIs only, no implementation
  - Abstract Classes – 1 or more methods unimplemented
  - Concrete Classes – All methods are implemented
- Inheritance (like Java/Objective C)
  - Interfaces may extend Interfaces
  - Classes extend no more than one Class
  - Classes can implement multiple Interfaces
- Only concrete classes can be instantiated

SIDL Grammar (3/3): Methods and Arguments

- Methods are public virtual by default
  - static methods are not associated with an object instance
  - final methods can not be overridden
- Arguments have 3 parts
  - Mode: can be in, out, or inout (like CORBA, but semantically different than F90)
  - Type: one of (bool, char, int, long, float, double, fcomplex, dcomplex, array<*Type*,*Dimension*>, enum, interface, class)
  - Name

Babelizing Legacy Code



1. Write your SIDL interface
2. Generate server side in your native langauge
3. Edit Implementation (Impls) to dispatch to your code (Do NOT modify the legacy library itself!)
4. Compile & Link into Library/DLL

CCA Common Component Architecture

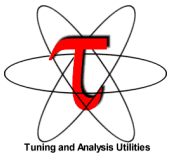
Known Projects Using Babel

(see www.llnl.gov/CASC/components/gallery.html for more)

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.

research.cs.vt.edu/lacsa



CCA Common Component Architecture

Babel & Legacy Code (e.g. MPI)

```
package mpi version 2.0 {
  class Comm {
     int send[Int]( in array<int,1,row-major> data,
                    in int dest, in int tag );
     ...
  }
}
```

mpi.sidl

CCA Common Component Architecture

Babel & Legacy Code (e.g. MPI)

```
struct mpi_Comm__data {
  /* DO-NOT-DELETE splicer.begin(mpi.Comm._data) */
  MPI_Comm com;
  /* DO-NOT-DELETE splicer.end(mpi.Comm._data) */
};
```

mpi_comm_Impl.h

```
int32_t
impl_mpi_Comm_sendInt( mpi_Comm self, SIDL_int__array data,
                       int32_t dest, int32_t tag ) {
  /* DO-NOT-DELETE splicer.begin(mpi.Comm.sendInt) */
  struct mpi_Comm__data *dptr = mpi_Comm__get_data( self );
  void * buff = (void*) SIDL_int__array_first(data);
  int count = length(data);
  return mpi_send( buff, count, MPI_INT, dest, tag, dptr->comm);
  /* DO-NOT-DELETE splicer.end(mpi.Comm.sendInt) */
}
```

mpi_comm_Impl.c

CCA Common Component Architecture

Investing in Babelization can improve the interface to the code.

"When Babelizing LEOS [an equation of state library at LLNL], I completely ignored the legacy interface and wrote the SIDL the way I thought the interface should be. After running Babel to generate the code, I found all the hooks I needed to connect LEOS without changing any of it. Now I've got a clean, new, object-oriented python interface to legacy code. Babel is doing much more than just wrapping here."

-- Charlie Crabb, LLNL (conversation)

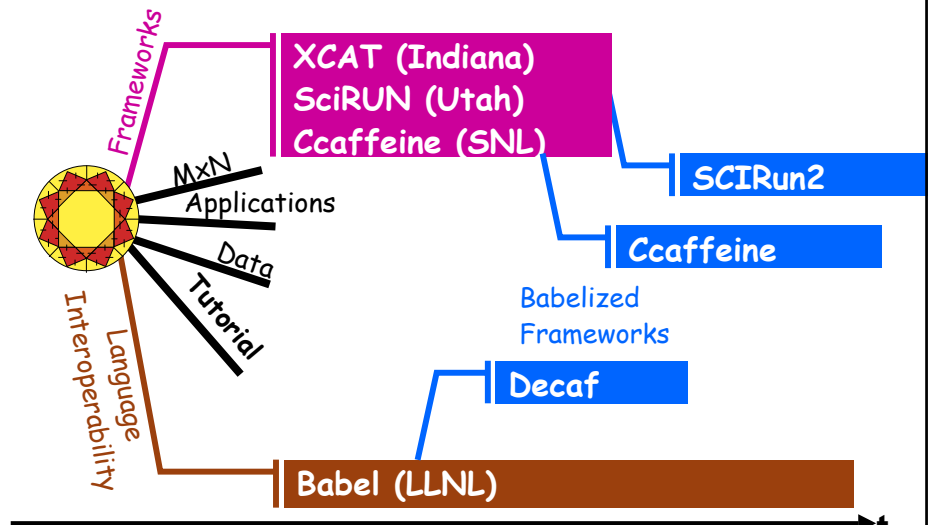
CCA Common Component Architecture

Babel Module's Outline

- Introduction
- Babel Basics
  - How to use Babel in a "Hello World" Example
  - SIDL Grammar
  - Example: Babel & Legacy Code
- Babel & CCA
  - Relationship between them
  - How to write a Babelized CCA Component

CCA Common Component Architecture

History of Babel & CCA



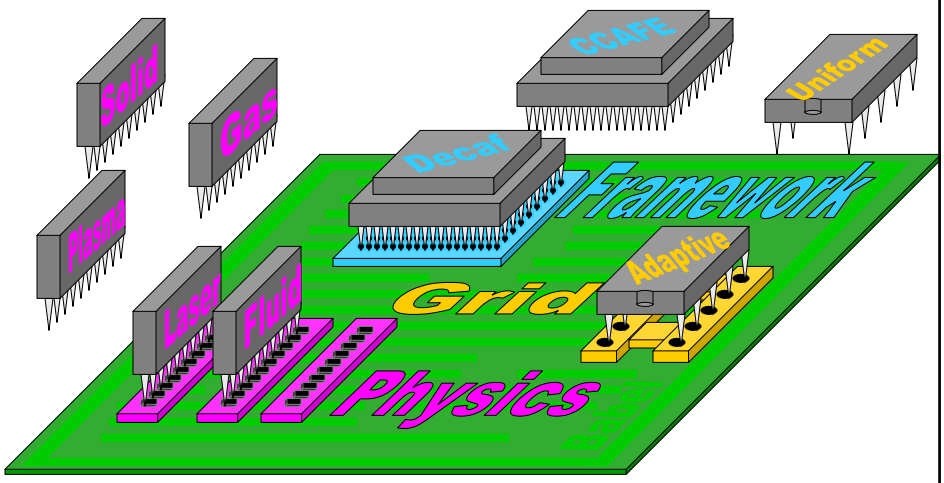
CCA Common Component Architecture

The CCA Spec is a SIDL File

```
package gov {
package cca version 0.6.2 {
    interface Port { }
    interface Component {
        void setServices( in Services svcs );
    }
    interface Services {
        Port getPort( in string portName );
        registerUsesPort( /*etc*/ );
        addProvidesPort( /*etc*/ );
    /*etc*/
```


CCA Common Component Architecture

The CCA from Babel's POV



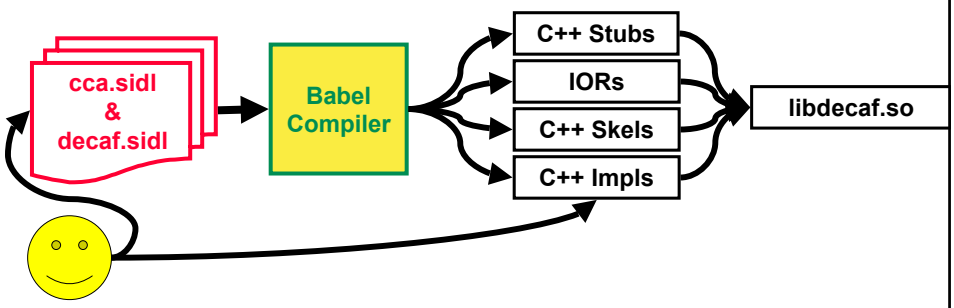
CCA Common Component Architecture

Decaf: Details & Disclaimers

- Babel is a hardened tool
- Decaf is an example, not a product
  - Distributed in “examples” subdirectory of Babel
  - Decaf has no GUI
- Decaf is CCA compliant
  - Babelized CCA Components can be loaded into Decaf, CCAFFEINE, and SCIRun2
- **“Understanding the CCA Specification Using Decaf”** **http://www.llnl.gov/CASC/components/docs/decaf.pdf**

CCA Common Component Architecture

How I Implemented Decaf



1. wrote decaf.sidl file
2. `babel --server=C++ cca.sidl decaf.sidl`
3. Add implementation details
4. Compile & Link into Library/DLL

CCA Common Component Architecture

How to Write and Use Babelized CCA Components

1. Define “Ports” in SIDL
2. Define “Components” that implement those Ports, again in SIDL
3. Use Babel to generate the glue-code
4. Write the guts of your component(s)

CCA Common Component Architecture

How to Write A Babelized CCA Component (1/3)

1. Define “Ports” in SIDL
   - CCA Port =
     - a SIDL Interface
     - extends gov.cca.Port

```
package functions version 1.0 {
    interface Function extends gov.cca.Port {
        double evaluate( in double x );
    }
}
```

CCA Common Component Architecture

How to Write A Babelized CCA Component (2/3)

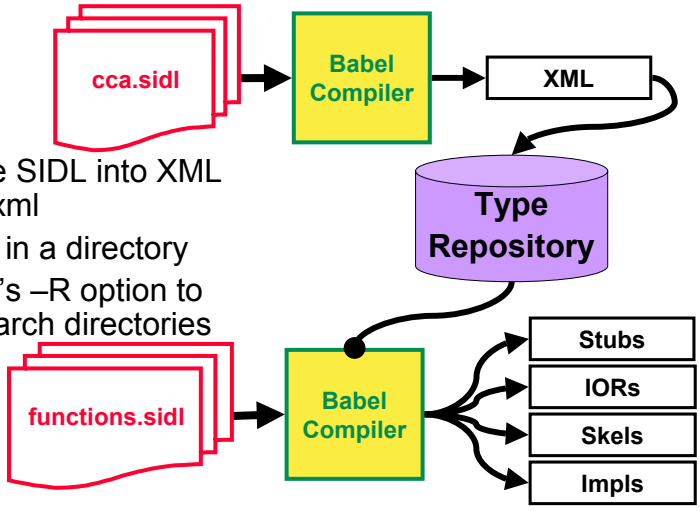
2. Define “Components” that implement those Ports
   - CCA Component =
     - SIDL Class
     - implements gov.cca.Component (& any provided ports)

```
class LinearFunction implements functions.Function,
                                gov.cca.Component {
      double evaluate( in double x );
      void setServices( in cca.Services svcs );
}
```

```
class LinearFunction implements-all
       functions.Function, gov.cca.Component { }
```


CCA Common Component Architecture

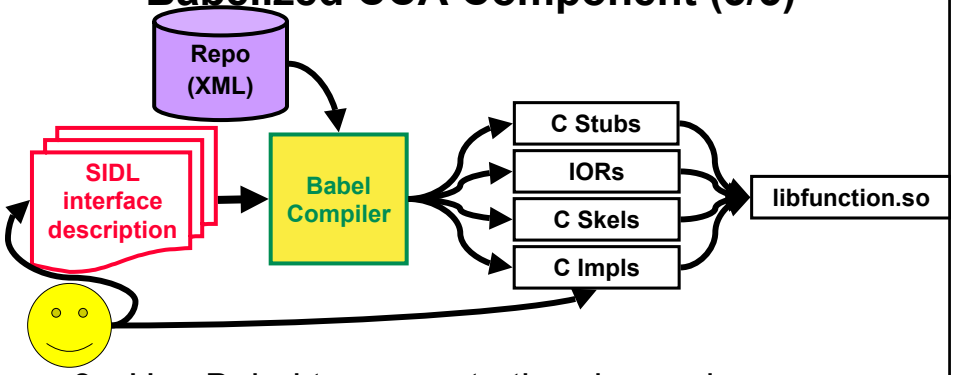
Tip: Use Babel’s XML output like precompiled headers in C++



1. precompile SIDL into XML
   --text=xml
2. store XML in a directory
3. Use Babel’s –R option to specify search directories

CCA Common Component Architecture

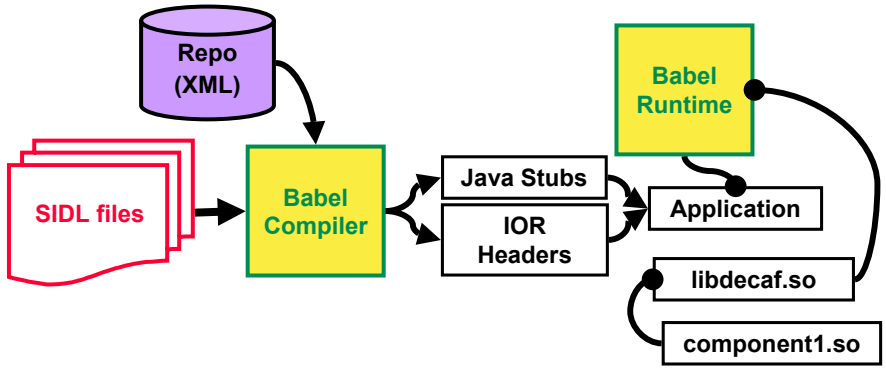
How to Write A Babelized CCA Component (3/3)



3. Use Babel to generate the glue code
   - `babel --server=C –Rrepo function.sidl`
4. Add implementation details

CCA Common Component Architecture

To Use the Decaf Framework



1. `babel --client=Java –Rrepo function.sidl`
2. Compile & Link generated Code & Runtime
3. Place DLLs in suitable location

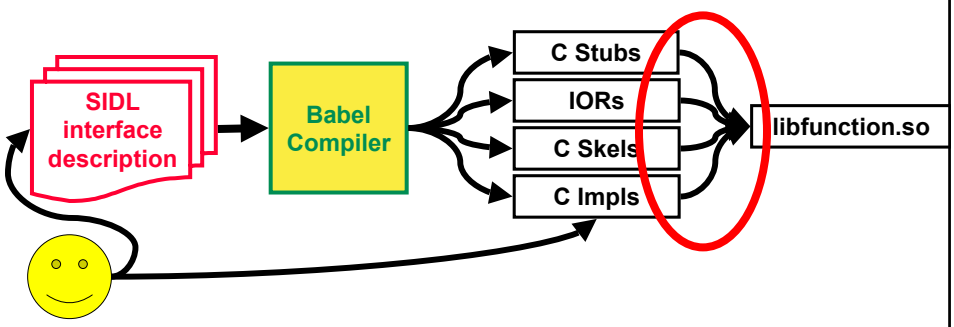
CCA Common Component Architecture

Limitations of Babel's Approach to Language Interoperabilty

- Babel is a code generator
  - Do obscure tricks no one would do by hand
  - Don't go beyond published language standards
- Customized compilers / linkers / loaders beyond our scope
  - E.g. icc and gcc currently don't mix on Linux
  - E.g. No C++-style templates in SIDL. (Would require special linkers/loaders to generate code for template instantiation, like C++ does.)
- Babel makes language interoperability feasible, but not trivial
  - Build tools severely underpowered for portable multi-language codes

CCA Common Component Architecture

What's the Hardest Part of this Process?



- Properly building libraries for multi-language use
- Dynamically loadable .so files are especially error prone
  - Not a lot of understanding or expertise in community
  - Causality chain between improperly constructed DLLs and observed bugs is often inscrutable and misleading

CCA Common Component Architecture

Summary

Legacy codes → Babelized CCA Components

- Reclassify your objects in your legacy code
  - Things customers create → CCA components
  - Logical groups of a component's functionality → CCA Port
  - Low level objects in your implementation → not exposed
- Generate SIDL File
  - CCA port → Babel Interface that extends the Babel interface called "gov.cca.Port"
  - CCA component → Babel Class that implements the Babel interface called "gov.cca.Component" (and possibly its "provides ports")
- Run Babel (choose server-language for your code)
- Articulate Impl files to dispatch to legacy code

CCA Common Component Architecture

Contact Info

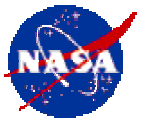
- Project: http://www.llnl.gov/CASC/components
  - Babel: language interoperability tool
  - Alexandria: component repository
  - Quorum: web-based parliamentary system
  - Gauntlet (coming soon): testing framework
- Bug Tracking: http://www-casc.llnl.gov/bugs
- Project Team Email: components@llnl.gov
- Mailing Lists: majordomo@lists.llnl.gov

  subscribe babel-users *[email address]*

  subscribe babel-announce *[email address]*

CCA
Common Component Architecture

A Simple CCA Component Application



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

CCA
Common Component Architecture

Goals

Show how CCA components are used to build an application to integrate numerically a continuous function using two different integration techniques

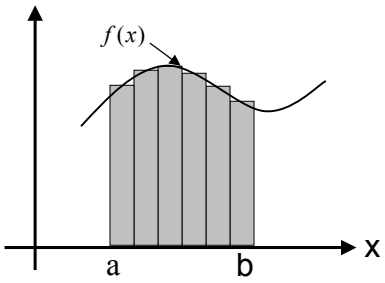
1. What the example does: the math
2. From math to components: the architecture
3. The making of components: inheritance and ports
4. Framework-component interactions
5. Putting it all together: the Ccaffeine way

CCA
Common Component Architecture

The Math: Integrator (1)

The midpoint numerical integrator

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\sum_{j=1}^{n} f\left(\frac{x_{j-1}+x_j}{2}\right)$$



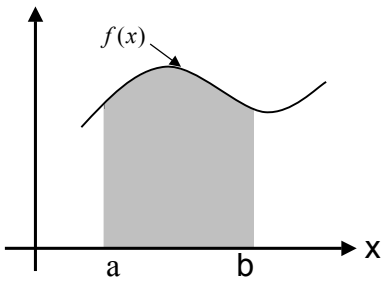
CCA
Common Component Architecture

The Math: Integrator (2)

The Monte Carlo integrator

$$\int_a^b f(x)dx \approx \frac{1}{b-a}\left(\frac{1}{N}\sum_{i=1}^{N} f(x_n)\right)$$

x_n Uniformally distributed in $[a,b]$



CCA Common Component Architecture

The math: Functions

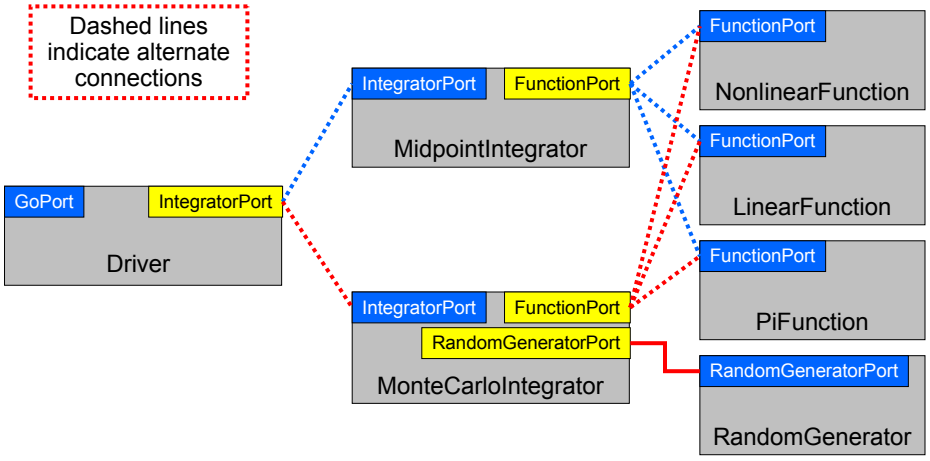
Linear Function $f_1(x) = 2x$

Nonlinear Function $f_2(x) = x^2$

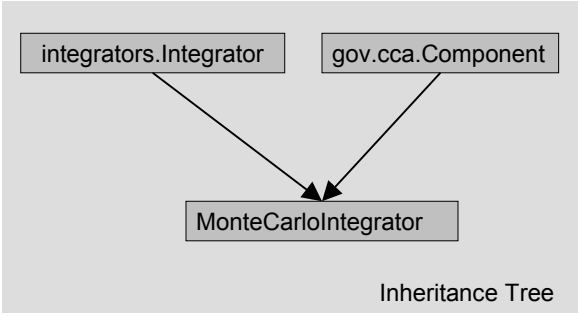
Pi Function $f_3(x) = \frac{4}{1+x^2}$

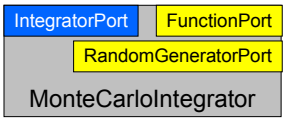
CCA Common Component Architecture

Available Components



The MonteCarloIntegrator Component





What makes it a component?
Inheritance from ***gov.cca.Component***

Where does the provided IntegratorPort come from?
Inheritance from ***integrators.Integrator***

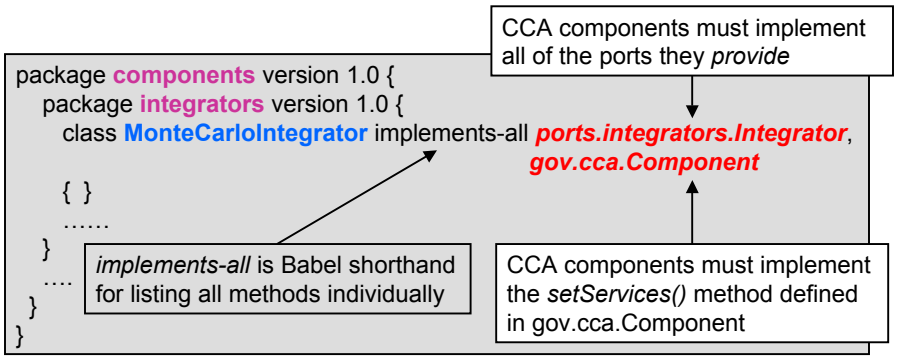
SIDL Definition of the Integrator Port

```
package example version 1.0 {
  package ports version 1.0 {
    package integrators version 1.0 {
      interface Integrator extends gov.cca.Port
      {
        double integrate(in double lowBound, in double upBound, in int count);
      }
    }
  .......
  }
```

CCA ports must inherit from gov.cca.Port, which contains no methods

Port definitions are used by:
- Components implementing (providing) the port
- Components using the port

SIDL Definition for a Component



Interactions Between Components and the Framework

- Framework-to-Component: ***setServices()***
  - Every CCA component must implement ***setServices()***
  - Called by framework after the component is instantiated.
  - Allows the component to tell the framework
    - Ports it provides
    - Ports it uses
  - Component should not acquire the port here – Reason: it may not be there yet !!!!

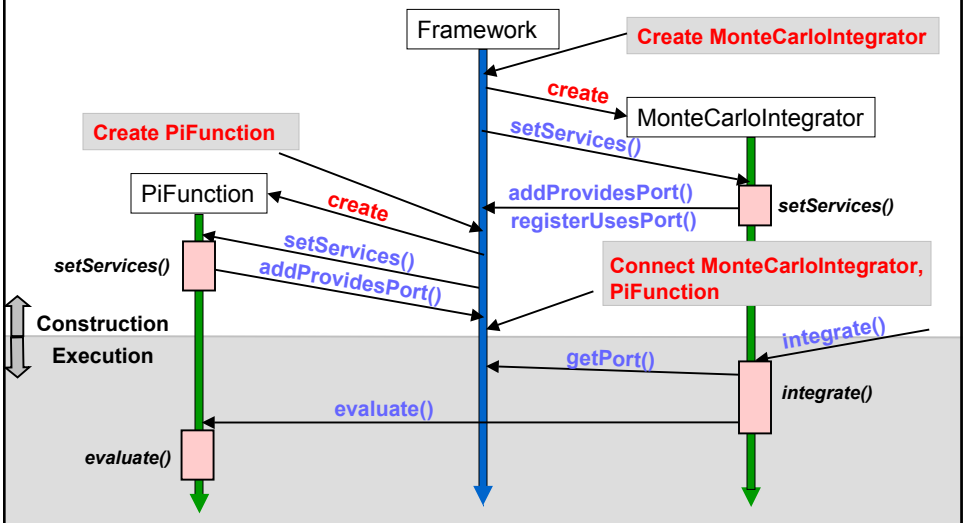
CCA Common Component Architecture

Component-to-Framework

- Mainly through ***Services*** object initially passed into ***setServices().***
- ***addProvidesPort()***, ***registerUsesPort()***:
  - Args: Component "*pointer*", PortName, PortType, PortProperties
  - Used in ***setServices(),*** and sometimes elsewhere, to tell framework what component will provide/use
- ***getPort(), releasePort()***
  - Called when component needs to actually invoke methods on another port
  - Matching using portType (not name).
- ***removeProvidesPort()***:
  - When all is done.

CCA Common Component Architecture

The Life Cycle Revisited



CCA Common Component Architecture

Actual Code for the MonteCarloIntegrator Component

The following slides illustrate the actual code for the component in C++

- ***setServices()*** method
- ***integrate()*** method
- ***.cca*** file (component metadata)

More examples to be shown in detail later

CCA Common Component Architecture

Example: *setservices()* in MonteCarloIntegrator (C++)

```
……
frameworkServices = services;
if (frameworkServices._not_nil ()) {
  gov::cca::TypeMap tm =  frameworkServices.createTypeMap ();
  gov::cca::Port p = self;
  frameworkServices.addProvidesPort (p,
                    "IntegratorPort",
                    "integrators.Integrator", tm);
// The Ports I use
  frameworkServices.registerUsesPort (
                    "FunctionPort",
                    "functions.Function", tm);
  frameworkServices.registerUsesPort (
                    "RandomGeneratorPort",
                    "randomgen.RandomGenerator", tm);
……
```

portName

portType

portProperties

CCA Common Component Architecture

Notes

- ***setServices()*** mainly used to inform the framework which ports the current component provides and/or uses.
- No actual connections between ports are established in ***setServices()***, since the "other" port may not yet exist !!!
- ***portName*** is unique per component.
- ***portType*** identifies the "*interface*" that the port implements (used to match user and provider).
- ***portProperties*** : list of port-specific key-value pairs.

CCA Common Component Architecture

Example: *integrate()* in MonteCarloIntegrator (C++)

```
……
 example::ports::functions::Function functionPort;
 example::ports::randomgen::RandomGenerator randomPort;
 double sum = 0.0;
 randomPort = frameworkServices.getPort ("RandomGeneratorPort");
 functionPort = frameworkServices.getPort ("FunctionPort");
 for (int i = 0; i < count; i++){
   double x = lowBound + (upBound - lowBound) *
                 randomPort.getRandomNumber();
   sum = sum + functionPort.evaluate(x);
 }
 frameworkServices.releasePort ("FunctionPort");
 frameworkServices.releasePort ("RandomGeneratorPort");
 return (upBound - lowBound) * sum / count;

……
```

CCA Common Component Architecture

Putting it all together

- Getting the application to do something:
  - Assembling the components into an application.
  - Launching the Application.
- Application assembly:
  - Framework need to be told what components to use, and where to find them.
  - Framework need to be told which ***uses*** port connects to which ***provides*** port.
- Application execution: the ***GO*** port:
  - Special ***provides*** port used to launch the application (after connections are established).
  - Has one method, ***go(),*** that is called by the framework to get the application going.

CCA Common Component Architecture

Oh Component , where art thou?

Component meta information

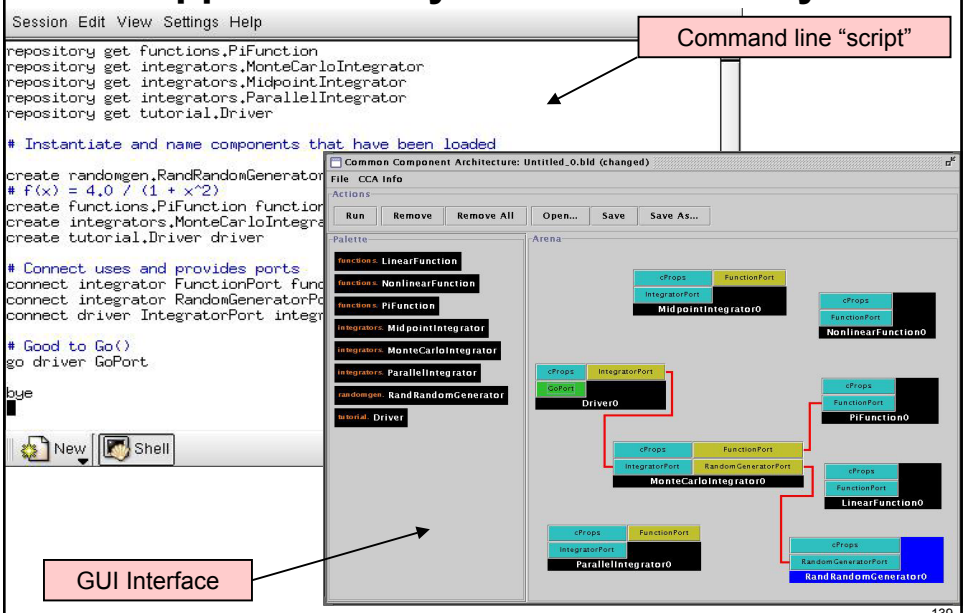
MonteCarloIntegrator.depl.cca

```
<componentDeployment
  name="example.components.integrators.MonteCarloIntegrator"
  uniqueID="norris@196.128.3.2#9.17.2003.dbg:/MonteCarloIntegrator"
  palletClassAlias="integrators_MonteCarlo">
  <environment>
    <ccaSpec binding="babel"/>
    <library loading="dynamic"
      name="libIntegrator-component-f90.so"
      location="/home/norris/cca/tutorial/src/components/integrators/f90/lib" />
  </environment>
</componentDeployment>
```

More details in the Ccaffeine Module

CCA
Common Component Architecture

App. Assembly The Ccafeine way



CCA
Common Component Architecture

Writing Components

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

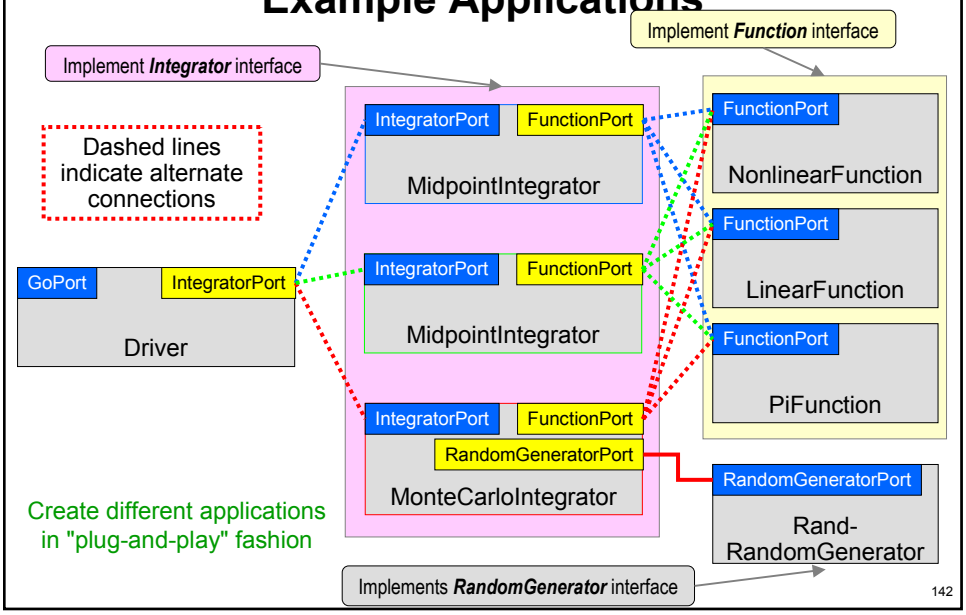
CCA Common Component Architecture

Module Overview

- Goal: present a step-by-step approach to designing and implementing CCA components
- Example application
- Steps involved in writing CCA components
  1. Interface definition; ports
  2. Defining SIDL packages
  3. Component implementation
     1. Framework interactions
     2. Component interactions: uses and provides ports
  4. Building

CCA Common Component Architecture

Example Applications



CCA Common Component Architecture

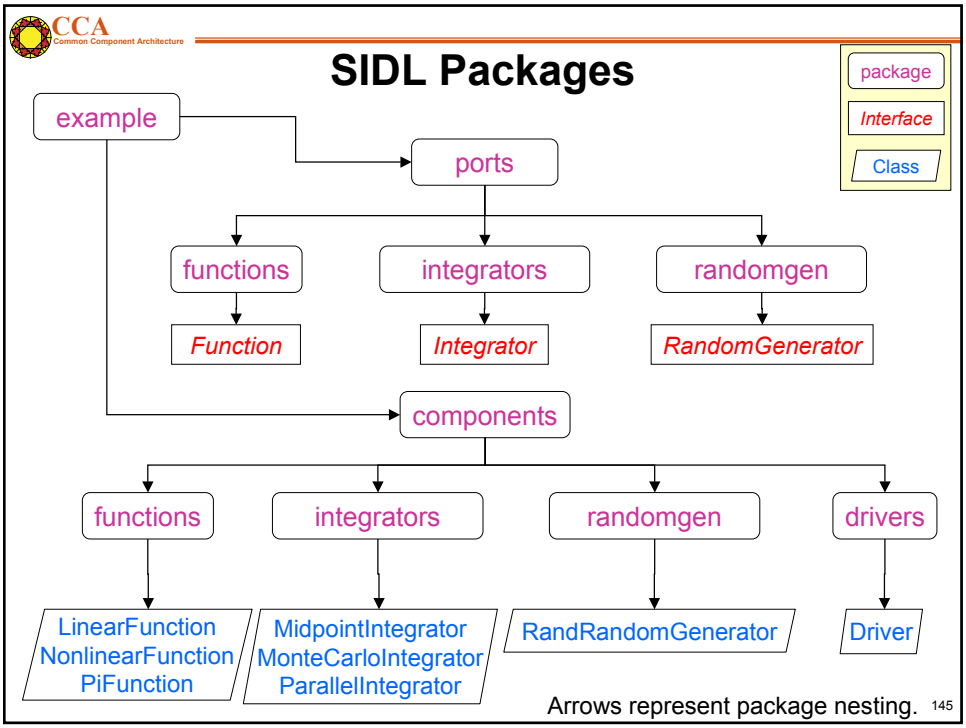
Port Definitions

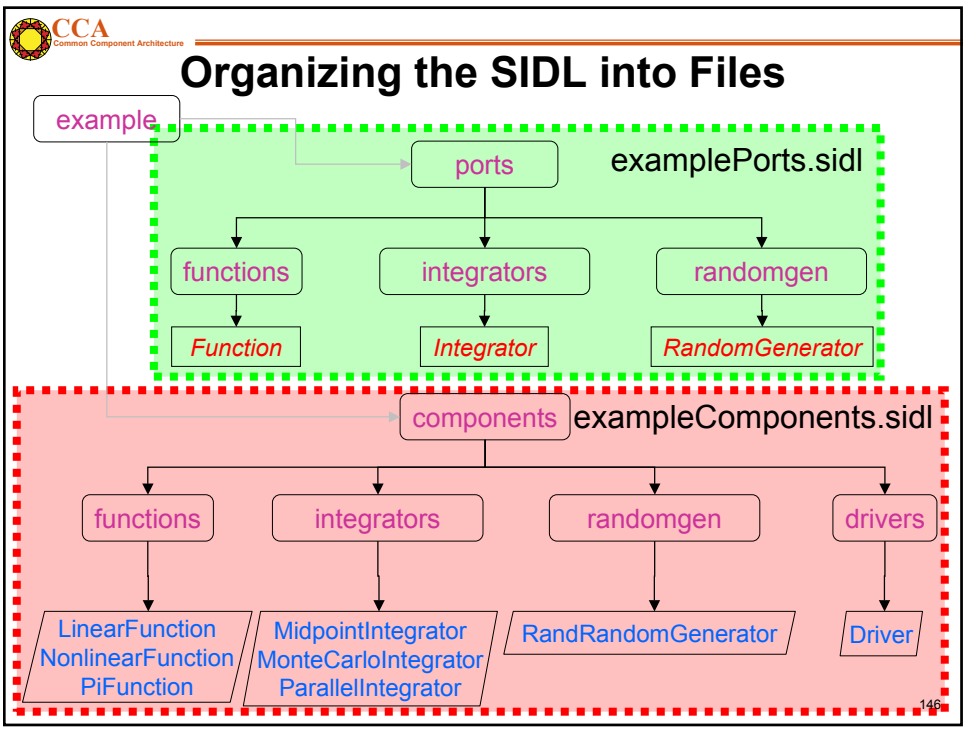
- Integrator
  - Computes the integral of a scalar function
- Random number generator
  - Generates a pseudo-random number
- Function
  - Computes a scalar function
- Go
  - Entry point into the application

CCA Common Component Architecture

Components

- Integrators (provides IntegratorPort, uses FunctionPort)
  - MonteCarloIntegrator (also uses RandomGeneratorPort)
  - MidpointIntegrator
  - ParallelIntegrator
- Functions (provides FunctionPort)
  - LinearFunction
  - NonlinearFunction
  - PiFunction
- Random number generators (provides RandomGeneratorPort)
  - RandRandomGenerator
- Driver (provides GoPort, uses IntegratorPort)


CCA
Common Component Architecture
SIDL Packages
package
Interface
Class
example
ports
functions
integrators
randomgen
Function
Integrator
RandomGenerator
components
functions
integrators
randomgen
drivers
LinearFunction
NonlinearFunction
PiFunction
MidpointIntegrator
MonteCarloIntegrator
ParallelIntegrator
RandRandomGenerator
Driver
Arrows represent package nesting.
145


CCA
Common Component Architecture
Organizing the SIDL into Files
example
ports
examplePorts.sidl
functions
integrators
randomgen
Function
Integrator
RandomGenerator
components
exampleComponents.sidl
functions
integrators
randomgen
drivers
LinearFunction
NonlinearFunction
PiFunction
MidpointIntegrator
MonteCarloIntegrator
ParallelIntegrator
RandRandomGenerator
Driver
146

CCA Common Component Architecture

examplePorts.sidl

```
package example version 1.0 {
  package ports version 1.0 {
    package functions version 1.0 {
      interface Function extends gov.cca.Port
      {
        double evaluate(in double x);
      }
    }
    package integrators version 1.0 {
      interface Integrator extends gov.cca.Port
      {
        double integrate(in double lowBound, in double upBound, in int count);
      }
    }
    package randomgen version 1.0 {
      interface RandomGenerator extends gov.cca.Port
      {
        double getRandomNumber();
      }
    }
  }
}
```


CCA Common Component Architecture

exampleComponents.sidl

```
package example version 1.0 {
  package components version 1.0 {
    package functions version 1.0 {
      class LinearFunction implements-all ports.functions.Function, gov.cca.Component
      { }
      class NonlinearFunction implements-all ports.functions.Function, gov.cca.Component
      { }
      class PiFunction implements-all ports.functions.Function, gov.cca.Component
      { }
    } // end package functions
    package integrators version 1.0 {
      class MonteCarloIntegrator implements-all ports.integrators.Integrator,
                                                gov.cca.Component
      { }
      class MidpointIntegrator implements-all ports.integrators.Integrator,
                                              gov.cca.Component
      { }
      class ParallelIntegrator implements-all ports.integrators.Integrator,
                                              gov.cca.Component
      { }
    } // end package integrators
```

exampleComponents.sidl (cont.)

```
  package randomgen version 1.0 {
    class RandRandomGenerator implements-all ports.randomgen.RandomGenerator,
                                             gov.cca.Component
    { }
  }

  // Driver component
  package drivers version 1.0 {
    class Driver implements-all gov.cca.ports.GoPort, gov.cca.Component
    { }
  }
 } // end package components
} // end package example
```


Generating Code with Babel

Goals:

- Generate implementation skeletons for example component classes *only*.
- Generate client stubs for interfaces and classes example classes implement or extend, e.g., CCA Port interface specification in cca.sidl.

```
> babel --text=XML --output-directory=repository cca.sidl \
        examplePorts.sidl exampleComponents.sidl
Babel: Parsing URL "file:/cca/tutorial/src/cca.sidl"...
Babel: Parsing URL "file:/cca/tutorial/src/examplePorts.sidl"...
Babel: Parsing URL "file:/cca/tutorial/src/exampleComponents.sidl"...
```

- The XML representation equivalent to the SIDL specification in *cca.sidl*, *examplePorts.sidl, and exampleComponents.sidl* was generated and stored in the *repository* directory.

Generating Code with Babel (Cont.)

```
> babel --client=C++ --repository-path=repository \
        --output-directory=cca-client/c++ gov.cca
Babel: Resolved symbol “gov.cca"...
```

Package name

- Using the XML repository in the *xml* directory for name resolution, the C++ **client** code for cca.sidl classes and interfaces is generated in the *cca-client/c++* directory.

```
> babel --client=C++ --repository-path=repository \
        --output-directory=ports/c++ example.ports
Babel: Resolved symbol "example.ports"...
> ls
cca-client/ cca.sidl components/ examplePorts.sidl
exampleComponents.sidl ports/ repository/
```

Package name

- Using the XML repository in the *xml* directory for name resolution, the C++ **client** code for the example.ports package interfaces is generated in the *ports/c++* directory.

Generating Code with Babel (Cont.)

```
> babel --server=C++ --repository-path=repository --hide-glue \
    -o components/functions/c++ example.components.functions
Babel: Resolved symbol "example.components.functions"...
```

- Using the XML repository in the *xml* directory for name resolution, the C++ **server** and client code for the example.components.functions package classes is generated in the *components/functions/c++* directory.

```
> babel --server=C++ --repository-path=repository --hide-glue \
        -o components/randomgen/c++ example.components.randomgen
> babel --server=C++ --repository-path=repository --hide-glue \
        -o components/drivers/c++ example.components.drivers
> babel --server=C++ --repository-path=repository --hide-glue \
        -o components/integrators/c++ \
        example.components.integrators.MidpointIntegrator \
        example.components.integrators.ParallelIntegrator
> babel --server=F90 --repository-path=repository --hide-glue \
        -o components/integrators/f90 \
        example.components.integrators.MonteCarloIntegrator
```

Resulting Directory Structure

```
components/drivers/c++/
                    |-- babel.make
                    |-- example_components_drivers_Driver_Impl.cc
                    |-- example_components_drivers_Driver_Impl.hh
                    `-- glue/
components/functions/c++/
                    |-- babel.make
                    |-- example_components_functions_LinearFunction_Impl.cc
                    |-- example_components_functions_LinearFunction_Impl.hh
                    |-- example_components_functions_NonlinearFunction_Impl.cc
                    |-- example_components_functions_NonlinearFunction_Impl.hh
                    |-- example_components_functions_PiFunction_Impl.cc
                    |-- example_components_functions_PiFunction_Impl.hh
                    `-- glue/
components/integrators/c++/
                    |-- babel.make
                    |-- example_components_integrators_MidpointIntegrator_Impl.cc
                    |-- example_components_integrators_MidpointIntegrator_Impl.hh
                    |-- example_components_integrators_ParallelIntegrator_Impl.cc
                    |-- example_components_integrators_ParallelIntegrator_Impl.hh
                    `-- glue/
components/integrators/f90/
                    |-- babel.make
                    |-- example_components_integrators_MonteCarloIntegrator_Impl.F90
                    |-- example_components_integrators_MonteCarloIntegrator _Mod.F90
                    `-- glue/
components/randomgen/c++/
                    |-- babel.make
                    |-- example_components_randomgen_RandRandomGenerator_Impl.cc
                    |-- example_components_randomgen_RandRandomGenerator_Impl.hh
                    `-- glue/
```


CCA Common Component Architecture

Component Implementation

- User code goes in *_Impl.* and/or *_Mod.* files, *always within splicer blocks:*

```
// DO-NOT-DELETE splicer.begin(...)
// Put additional inheritance here...
// DO-NOT-DELETE splicer.end(...)
```

File: components/integrators/c++/example_components_integrators_MidpointIntegrator_Impl.hh

MidpointIntegrator Component: Header File Fragment (C++)

```
namespace example {
 namespace components {
  namespace integrators {

   /**
    * Symbol "example.components.integrators.MidpointIntegrator" (version 1.0)
    */
   class MidpointIntegrator_impl
   // DO-NOT-DELETE splicer.begin(example.components.integrators.MidpointIntegrator._inherits)
   // Put additional inheritance here...
   // DO-NOT-DELETE splicer.end(example.components.integrators.MidpointIntegrator._inherits)
   {

   private:
    // Pointer back to IOR.
    // Use this to dispatch back through IOR vtable.
    MidpointIntegrator self;

    // DO-NOT-DELETE splicer.begin(example.components.integrators.MidpointIntegrator._implementation)
    // Put additional implementation details here...
    gov::cca::Services  frameworkServices;
    // DO-NOT-DELETE splicer.end(example.components.integrators.MidpointIntegrator._implementation)

... more Babel-generated code ...
```

File generated by Babel.
***One line** added by programmer.*

File: components/integrators/c++/example_components_integrators_MidpointIntegrator_Impl.cc

MidpointIntegrator Impl file fragment (C++)

```
// user defined constructor
void example::components::integrators::MidpointIntegrator_impl::_ctor() {
  // DO-NOT-DELETE splicer.begin(example.components.integrators.MidpointIntegrator._ctor)
  // add construction details here
  // DO-NOT-DELETE splicer.end(example.components.integrators.MidpointIntegrator._ctor)
}
// user defined destructor
void example::components::integrators::MidpointIntegrator_impl::_dtor() {
  // DO-NOT-DELETE splicer.begin(example.components.integrators.MidpointIntegrator._dtor)
  // add destruction details here
  // DO-NOT-DELETE splicer.end(example.components.integrators.MidpointIntegrator._dtor)
}
void example::components::integrators::MidpointIntegrator_impl::setServices (
  /*in*/ gov::cca::Services services ) throw ()
{
  // DO-NOT-DELETE splicer.begin(example.components.integrators.MidpointIntegrator.setServices)
  // insert implementation here
  // DO-NOT-DELETE splicer.end(example.components.integrators.MidpointIntegrator.setServices)
}
Double example::components::integrators::MidpointIntegrator_impl::integrate (
  /*in*/ double lowBound, /*in*/ double upBound, /*in*/ int32_t count )  throw ()
{
  // DO-NOT-DELETE splicer.begin(example.components.integ
  // insert implementation here
  // DO-NOT-DELETE splicer.end(example.components.integrators.MidpointIntegrator.integrate)
}
```

As originally generated by Babel, before modified by programmer

File: components/integrators/c++/example_components_integrators_MidpointIntegrator_Impl.cc

MidpointIntegrator Component: Framework Interaction (C++)

```
example::components::integrators::MidpointIntegrator_impl::setServices (
 /*in*/ gov::cca::Services services )
throw ()
{
 // DO-NOT-DELETE splicer.begin(example.components.integrators.MidpointIntegrator.setServices)
 frameworkServices = services;
 if (frameworkServices._not_nil ()) {
    gov::cca::TypeMap tm = frameworkServices.createTypeMap ();
    gov::cca::Port p = self;    // Babel required cast

    // Port provided by all Integrator implementations
    frameworkServices.addProvidesPort (p, "IntegratorPort", "integrators.Integrator", tm);

    // Ports used by MonteCarloIntegrator
    frameworkServices.registerUsesPort ("FunctionPort", "functions.Function",
                                        tm);
 }
 // DO-NOT-DELETE splicer.end(example.components.integrators.MidpointIntegrator.setServices)
}
```

Save a reference to the framework's Services object

Port name

Port type

File: components/integrators/c++/example_components_integrators_ParallelIntegrator_Impl.cc

ParallelIntegrator integrate() Method (C++)

```
double
example::components::integrators::ParallelIntegrator_impl::integrate(/*in*/ double lowBound,
                        /*in*/ double upBound, /*in*/ int32_t count ) throw ()
{
 // DO-NOT-DELETE
splicer.begin(example.components.integrators.ParallelIntegrator.integrate)
  gov::cca::Port port;
  example::ports::functions::Function function_port;

  // Get Function port
  function_port = frameworkServices.getPort("FunctionPort");

  int n, myid, numprocs, i;
  double result, myresult, h, sum, x;
  int namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Get_processor_name(processor_name, &namelen);

  fprintf(stderr, "Process %d on %s: number of intervals = %d\n", myid,
        processor_name, count);
  fflush(stderr);
  // ... Continued on next page...
```

Based on MidpointIntegrator

Get a Function reference

Parallel environment details

File: components/integrators/c++/example_components_integrators_ParallelIntegrator_Impl.cc

ParallelIntegrator integrate() Method (Cont.)

```
// ...
  MPI_Bcast(&count, 1, MPI_INT, 0, MPI_COMM_WORLD);
  if (count == 0) {
    return -1;
  } else {
    h = (upBound - lowBound) / (double) count;
    sum = 0.0;
    for (i = myid + 1; i <= count; i += numprocs) {
      x = h * ((double) i - 0.5);
      sum += function_port.evaluate(x);
    }
    myresult = h * sum;

    MPI_Reduce(&myresult, &result, 1, MPI_DOUBLE, MPI_SUM, 0,
               MPI_COMM_WORLD);
  }

  frameworkServices.releasePort("FunctionPort");
  printf("result is %f\n", result);
  return result;

 // DO-NOT-DELETE splicer.end(example.components.integrators.ParallelIntegrator.integrate)
}
```

Evaluate function

Compute integral in parallel

Release port

Return integral value

A Fortran Integrator Implementation: MonteCarloIntegrator

- Babel code generation
  - Port (client):

```
> babel --client=F90 --repository-path=repository -o ports/f90 \
        example.ports.integrators.Integrator
```

  - Component (server):

```
> babel --server=F90 --repository-path=repository --hide-glue \
        -o components/integrators/f90 \
        example.components.integrators.MonteCarloIntegrator
```

File: components/integrators/f90/example_components_integrators_MonteCarloIntegrator_Mod.F90

MonteCarloIntegrator Component: Module File (F90)

```
#include"example_components_integrators_MonteCarloIntegrator_fAbbrev.h"
module example_components_integrators_MonteCarloIntegrator_impl

! DO-NOT-DELETE splicer.begin(example.components.integrators.MonteCarloIntegrator.use)
! Insert use statements here...
! Framework Services module
use gov_cca_Services
! DO-NOT-DELETE splicer.end(example.components.integrators.MonteCarloIntegrator.use)

type example_components_integrators_MonteCarloIntegrator_private
  sequence
  ! DO-NOT-DELETE splicer.begin(example.components.integrators.MonteCarloIntegrator.private_data)
  ! integer :: place_holder ! replace with your private data
  type(gov_cca_Services_t) :: frameworkServices
  ! DO-NOT-DELETE splicer.end(example.components.integrators.MonteCarloIntegrator.private_data)
end type example_components_integrators_MonteCarloIntegrator_private

type example_components_integrators_MonteCarloIntegrator_wrap
  sequence
  type(example_components_integrators_MonteCarloIntegrator_private), pointer :: d_private_data
end type example_components_integrators_MonteCarloIntegrator_wrap

end module example_components_integrators_MonteCarloIntegrator_impl
```

Framework Services object handle

File: components/integrators/f90/example_components_integrators_MonteCarloIntegrator_Impl.F90

MonteCarloIntegrator Component: Framework Interaction (F90)

```
recursive subroutine MonteC_setServicesucff4xebul_mi(self, services)
 use example_components_integrators_MonteCarloIntegrator
 use gov_cca_Services
 use example_components_integrators_MonteCarloIntegrator_impl
 ! DO-NOT-DELETE splicer.begin(example.components.integrators.MonteCarloIntegrator.setServices.use)
 ! Insert use statements here...
 use gov_cca_TypeMap
 use gov_cca_Port
 use SIDL_BaseException
 ! DO-NOT-DELETE splicer.end(example.components.integrators.MonteCarloIntegrator.setServices.use)
 implicit none
 type(example_components_integrators_MonteCarloIntegrator_t) :: self
 type(gov_cca_Services_t) :: services
 ! DO-NOT-DELETE splicer.begin(example.components.integrators.MonteCarloIntegrator.setServices)
 type(gov_cca_TypeMap_t)          :: myTypeMap
 type(gov_cca_Port_t)             :: integratorPort
 type(SIDL_BaseException_t)       :: excpt
 ! Access private data
 type(example_components_integrators_MonteCarloIntegrator_wrap) :: pd
 external example_components_integrators_MonteCarloIntegrator__get_data_m
 call example_components_integrators_MonteCarloIntegrator__get_data_m(self, pd)
 ! Set my reference to the services handle
 pd%d_private_data%frameworkServices = services
 call addRef(services)
 ! Create a TypeMap with my properties
 call createTypeMap(pd%d_private_data%frameworkServices, myTypeMap, excpt)
```

Save a handle to the Services object

MonteCarloIntegrator Component: Framework Interaction (Continued)

```
call cast(self, integratorPort)

! Register my provides port
call addProvidesPort(pd%d_private_data%frameworkServices, integratorPort, &
                     'IntegratorPort', 'integrators.Integrator', &
                     myTypeMap, excpt)
if (not_null(excpt)) then
  write(*, *) 'Exception: MonteCarloIntegratory:setServices addProvidesPort'
end if

! The ports I use
call registerUsesPort(pd%d_private_data%frameworkServices, &
                      'FunctionPort', 'functions.Function', &
                      myTypeMap, excpt)

call registerUsesPort(pd%d_private_data%frameworkServices, &
                      'RandomGeneratorPort', 'ramdomgen.RandomGenerator', &
                      myTypeMap, excpt)

call deleteRef(myTypeMap)

! DO-NOT-DELETE splicer.end(example.components.integrators.MonteCarloIntegrator.setServices)
end subroutine MonteC_setServicesucff4xebul_mi
```

Explicit cast to Port

TypeMap

Exception

Port type

Port name

Port type

MonteCarloIntegrator Component: integrate() Method (F90)

```
recursive subroutine MonteCar_integrateni9nnumrzd_mi(self, lowBound, upBound,  &
 count, retval)
 use example_components_integrators_MonteCarloIntegrator
 use example_components_integrators_MonteCarloIntegrator_impl
 ! DO-NOT-DELETE splicer.begin(example.components.integrators.MonteCarloIntegrator.integrate.use)
 ! Insert use statements here...
 use example_ports_functions_Function
 use example_ports_randomgen_RandomGenerator
 use gov_cca_Services
 use gov_cca_Port
 use sidl_BaseException
 ! DO-NOT-DELETE splicer.end(example.components.integrators.MonteCarloIntegrator.integrate.use)
 implicit none
 type(example_components_integrators_MonteCarloIntegrator_t) :: self
 real (selected_real_kind(15, 307)) :: lowBound
 real (selected_real_kind(15, 307)) :: upBound
 integer (selected_int_kind(9)) :: count
 real (selected_real_kind(15, 307)) :: retval

! DO-NOT-DELETE splicer.begin(example.components.integrators.MonteCarloIntegrator.integrate)
! Insert the implementation here...
```

MonteCarloIntegrator Component: integrate() Method (Cont.)

```
! DO-NOT-DELETE splicer.begin(example.components.integrators.MonteCarloIntegrator.integrate)
! Insert the implementation here...
  type(gov_cca_Port_t) :: generalPort
  type(example_ports_functions_Function_t) :: functionPort
  type(example_ports_randomgen_RandomGenerator_t) :: randomPort
  type(SIDL_BaseException_t) :: excpt
  type(example_components_integrators_MonteCarloIntegrator_wrap) :: pd
  external example_components_integrators_MonteCarloIntegrator__get_data_m
  real (selected_real_kind(15, 307)) :: sum, width, x, func
  integer (selected_int_kind(9)) :: i
  ! Access private data
  call example_components_integrators_MonteCarloIntegrator__get_data_m(self, pd)
  retval = -1
  if (not_null(pd%d_private_data%frameworkServices)) then
    ! Obtain a handle to a FunctionPort
    call getPort(pd%d_private_data%frameworkServices, 'FunctionPort', generalPort, excpt)
    call cast(generalPort, functionPort)
    ! Obtain a handle to a RandomGeneratorPort
    call getPort(pd%d_private_data%frameworkServices, 'RandomGeneratorPort', generalPort, excpt)
    call cast(generalPort, randomPort)
    ! Compute integral
    sum = 0
    width = upBound - lowBound
    do i = 1, count
      call getRandomNumber(randomPort, x)
      x = lowBound + width*x
      call evaluate(functionPort, x, func)
      sum = sum + func
    enddo
    retval = width*sum/count
    call deleteRef(generalPort)
    call deleteRef(randomPort)
    call releasePort(pd%d_private_data%frameworkServices, 'RandomGeneratorPort', excpt)
    call deleteRef(functionPort)
    call releasePort(pd%d_private_data%frameworkServices, 'FunctionPort', excpt)
  endif   ! end of implementation
```

Access component's data

Get a Function reference

Get a RandomGenerator reference

Return integral value

Release ports

Building components

- Dynamic (shared) libraries
  - For each port, build a dynamic library of the client code for each supported language
  - For each component or a set of components, build a dynamic library
    - Babel components and Ccaffeine: build a shared library for the implementation (server). No linking required of libraries for implementations of components on which current component depends; instead, link to the *client* libraries for the ports used and provided.
    - Non-component libraries on which a component depends directly (e.g., BLAS), must be linked explicitly when the shared library is created

Makefile for MidpointIntegrator (C++)

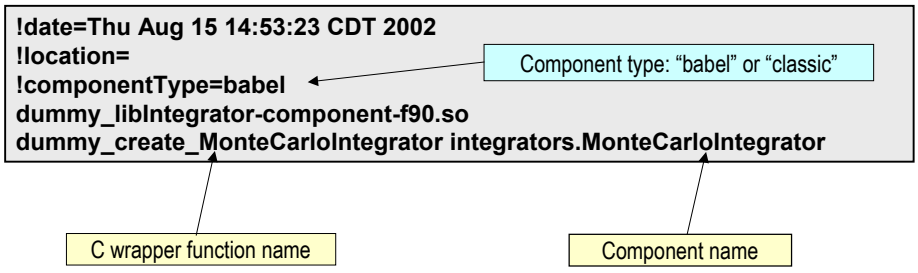
```
# File: components/integrators/c++/Makefile
include ../../../../Makefile.Vars
include babel.make
include glue/babel.make
VPATH = glue
INCLUDES = -I$(BABEL_ROOT)/include  -I. \
        -I$(CCATUT_SIDL_ROOT)/ports/c++/include
all: libIntegrator-component-c++.so
.c.o:
        gcc -g -fPIC $(INCLUDES) -c $< -o $(<:.c=.o)
.cc.o:
        g++ -g -fPIC $(INCLUDES) -c $< -o $(<:.cc=.o)
IMPLOBJS = $(IMPLSRCS:.cc=.o)
GLUEOBJS = glue/$(IORSRCS:.c=.o) glue/$(SKELSRCS:.cc=.o) \
        glue/$(STUBSRCS:.cc=.o)
OBJS = $(IMPLOBJS) $(GLUEOBJS)
LIBS = -Wl,-rpath,$(BABEL_ROOT)/lib -L$(BABEL_ROOT)/lib -lsidl  \
        -L$(CCATUT_SIDL_ROOT)/ports/c++/lib -lfunction-port-c++ \
        -L$(CCATUT_SIDL_ROOT)/ports/c++/lib -lintegrator-port-c++ \
       -L$(CCATUT_SIDL_ROOT)/cca-client/c++ -lcca-client-c++

libIntegrator-component-c++.so: $(OBJS)
        g++ -shared $(INCLUDES) $(IMPLOBJS) glue/*.o -o $@  $(LIBS)
clean:
        $(RM) *.o glue/*.o libIntegrator-component-c++.so
```


MonteCarloIntegrator: integrators.depl.cca

```
<componentDeployment
  name="example.components.integrators.MonteCarloIntegrator"
  uniqueID="norris@196.128.3.2#9.17.2003.dbg:/MonteCarloIntegrator"
  palletClassAlias="integrators_MonteCarlo">
  <environment>
    <ccaSpec binding="babel"/>
    <library loading="dynamic"
       name="libIntegrator-component-f90.so"
       location="/home/norris/cca/tutorial/src/components/integrators/f90/lib" />
  </environment>
</componentDeployment>
```

MonteCarloIntegrator: integrators.cca (soon to be deprecated)

- Ccaffeine-specific file giving the type of component (e.g., “babel”), name of the dynamic library, and creation method for each component.



Additional Examples

Other Component Implementations

- MidpointIntegrator: C++
- MonteCarloIntegrator: F90
- RandRandomGenerator: C++
- PiFunction: C++
- Driver: C++

File: components/randomgen/c++/example_components_randomgen_RandRandomGenerator_Impl.hh

RandRandomGenerator Component: C++ Implementation Header Fragment

```
namespace example {
  namespace components {
    namespace randomgen {
      /**
       * Symbol "example.components.randomgen.RandRandomGenerator" (version 1.0)
       */
      class RandRandomGenerator_impl
      {
      private:
        // Pointer back to IOR.
        // Use this to dispatch back through IOR vtable.
        RandRandomGenerator self;

        // DO-NOT-DELETE splicer.begin(example.components.randomgen.RandRandomGenerator._implementation)
        // Put additional implementation details here...
        gov::cca::Services  frameworkServices;
        // DO-NOT-DELETE splicer.end(example.components.randomgen.RandRandomGenerator._implementation)

        ...
      }; // end class RandRandomGenerator_impl
    } // end namespace randomgen
  } // end namespace components
} // end namespace example
```

Reference to framework Services object

File: components/randomgen/c++/example_components_randomgen_RandRandomGenerator_Impl.cc

RandRandomGenerator Component (C++): getRandomNumber() Implementation

```
/**
 * Method:  getRandomNumber[]
 */
double
examples::components::randomgen::RandRandomGenerator_impl::getRandomNumber ()
throw ()

{
  // DO-NOT-DELETE splicer.begin(example.components.randomgen.RandRandomGenerator.getRandomNumber)
  // insert implementation here
  double random_value = static_cast < double >(rand ());
  return random_value / RAND_MAX;

  // DO-NOT-DELETE splicer.end(example.components.randomgen.RandRandomGenerator.getRandomNumber)
}
```


File: components/functions/c++/example_components_functions_PiFunction_Impl.hh

PiFunction Component (C++): Impl. Header Fragment

```
namespace example {
  namespace components {
    namespace functions {

      /**
       * Symbol "example.components.functions.PiFunction" (version 1.0)
       */
      class PiFunction_impl
      {
      private:
        // Pointer back to IOR.
        // Use this to dispatch back through IOR vtable.
        PiFunction self;

        // DO-NOT-DELETE splicer.begin(functions.PiFunction._implementation)
        // Put additional implementation details here...
        gov::cca::Services  frameworkServices;
        // DO-NOT-DELETE splicer.end(functions.PiFunction._implementation)
        ...
      }; // end class PiFunction_impl
    } // end namespace functions
  } // end namespace components
} // end namespace example
```

PiFunction Component (C++): evaluate() Method

```
/**
 * Method:  evaluate[]
 */
double
example::components::functions::PiFunction_impl::evaluate (  /*in*/ double x )
throw ()
{
  // DO-NOT-DELETE splicer.begin(example.components.functions.PiFunction.evaluate)
  // insert implementation here

   return 4.0 / (1.0 + x * x);

  // DO-NOT-DELETE splicer.end(example.components.functions.PiFunction.evaluate)
}
```


Driver Component (C++): Framework Interaction

```
tutorial::Driver_impl::setServices (  /*in*/ gov::cca::Services services )
throw ()
{
  // DO-NOT-DELETE splicer.begin(example.components.drivers.Driver.setServices)
  frameworkServices = services;
  if (frameworkServices._not_nil ()) {
     gov::cca::TypeMap tm = frameworkServices.createTypeMap ();

     gov::cca::Port p = self;     // Babel-required cast

     // Port provided by Function implementations
     frameworkServices.addProvidesPort (p, "GoPort",
                                        "gov.cca.ports.GoPort", tm);

     // Port used by the Driver component
     frameworkServices.registerUsesPort ("IntegratorPort",
                                        "integrators.Integrator", tm);
   }
   // DO-NOT-DELETE splicer.end(example.components.drivers.Driver.setServices)
}
```

Driver Component (C++): GoPort implementation

```
int32_t
tutorial::Driver_impl::go () throw ()
{
 // DO-NOT-DELETE splicer.begin(example.components.drivers.Driver.go)
 double value;
 int count = 100000;   // number of intervals/random samples
 double lowerBound = 0.0, upperBound = 1.0;

 // Ports
 gov::cca::Port port;
 example::ports::integrators::Integrator integrator;

 port = frameworkServices.getPort("IntegratorPort");
 integrator = port;

 value = integrator.integrate (lowerBound, upperBound, count);

 fprintf(stdout,"Value = %lf\n", value);

 frameworkServices.releasePort ("IntegratorPort");
 return 0;

 // DO-NOT-DELETE splicer.end(example.components.drivers.Driver.go)
}
```

Get an Integrator port

Invoke the integrate method

Output integration result

Release ports

CCA Common Component Architecture

Introduction to the Ccaffeine Framework

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

CCA Common Component Architecture

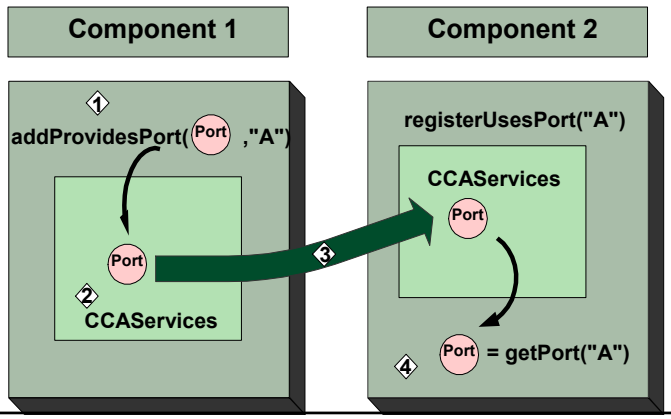
Outline

- What is a CCA Framework and what is Ccaffeine?
- How can I slip my own component into Ccaffeine?
- How do I run Ccaffeine?
- Parallel components using Ccaffeine and MPI.
- Live Demo – how does it work?

CCA Common Component Architecture

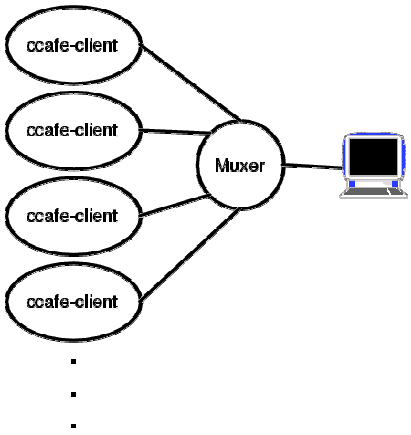
CCA What CCA compliant framework is expected to do …

- Exchange interfaces among components without one needing to know more about the other than the interface itself



Interactive Parallel Components: what Ccaffeine does

- Executable ccafe-client:
  - PVM, MPI, or whatever is used for communication between clients
  - Muxer enforces “single process image” of SPMD parallel computing
- How To:
  - Build Ccaffeine
  - Run Ccaffeine



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

How to build Ccaffeine

- Have a look at http://www.cca-forum.org/ccafe
  1. Obtain the required packages
     - gcc (http://gcc.gnu.org)
     - Java (>jdk1.2) (http://java.sun.com)
     - MPI (http://www.mcs.anl.gov/mpi/mpich)
     - BOOST headers (http://www.boost.org)
     - Babel (http://www.llnl.gov/casc/components/babel.html)
     - Ccaffeine tar ball download
     - Optional software
       - Fortran 77 and 90 compilers
       - Ruby
       - Python 2.x
  2. Install prerequisites

RPMs available for RedHat Linux, but not guaranteed to be in sync with tutorial source code available on web

CCA Common Component Architecture

How to build Ccaffeine (cont'd)

- Untar Ccaffeine-xxx.tgz in build dir
  - 3 directories appear cca-spec-babel (*the* spec), cca-spec-classic (old C++ spec), dccafe
- Run configure
  - If confused type "configure --help"; example options:

(cd ./cca-spec-babel; configure --with-babel=/usr/local/babel \
--with-jdk12=/usr/local/java;make; make install)

(cd ./cca-spec-classic; configure; make; make install)

(cd ./dccafe; ./configure --with-cca-babel=`pwd`/../cca-spec-babel \
--with-cca-classic=`pwd`/../cca-spec-classic –with-babel=/usr/local/babel-0.8.4 \
--with-mpi=/usr/local/mpich --with-jdk12=/usr/local/java \
--with-lapack=/home/rob/cca/dccafe/../LAPACK/liblapack.a \
--with-blas=/home/rob/cca/dccafe/../LAPACK/libblas.a; make; make install)

CCA Common Component Architecture

Ccaffeine build (cont'd)

- Example output at "make install" completion:

```
================================================================
                Testing the Ccaffeine build ...
proceeding with env vars:
# LD_LIBRARY_PATH=/home/norris/cca/dccafe/cxx/dc/babel/babel-
   cca/server:/home/software/mpich-1.2.5-
   ifc/lib/shared:/home/norris/cca/babel-
   0.8.4/lib:/usr/local/lib/python2.2/config:/usr/local/intel/compiler70/
   ia32/lib:/usr/local/lib:/usr/local/lib
# SIDL_DLL_PATH=/home/norris/cca/dccafe/lib
didn't crash or hang up early ...  looks like it is working.
Looks like CLASSIC dccafe is working.
Looks like BABEL dccafe is working.
done with Ccaffeine tests.
simpleTests: output is in
   /home/norris/cca/dccafe/simpleTests.out.XXXAL8Cmk.
================================================================
```

Note: depending on environment settings, sometimes the simple tests may fail but you may still have a functional framework.

CCA Common Component Architecture

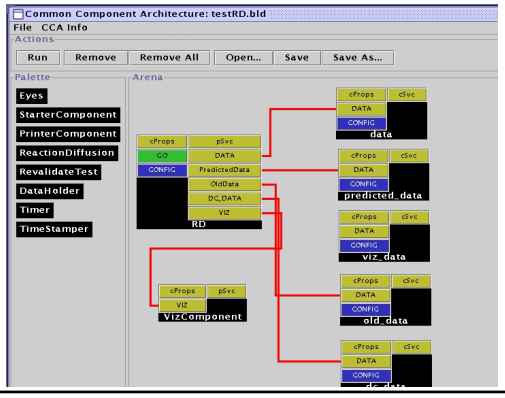
Running Ccaffeine

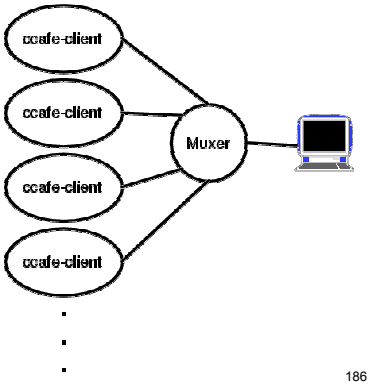
- Framework needs to be told:
  - Where to find components
  - Which components to instantiate
  - Which **uses** port gets connected to which **provides** port
  - Which **go** port sets the application in motion
- User-Ccaffeine interaction techniques:
  - GUI interface (with some Ccaffeine scripting help)
  - Pure Ccaffeine scripting (useful in batch mode)
  - Python component driver (with some Ccaffeine scripting help)

CCA Common Component Architecture

How to run Ccaffeine:

- Ccaffeine interactive language
  - Used to configure batch and interactive sessions
  - Allows useful “defaults”
  - Allows the GUI to talk over a socket



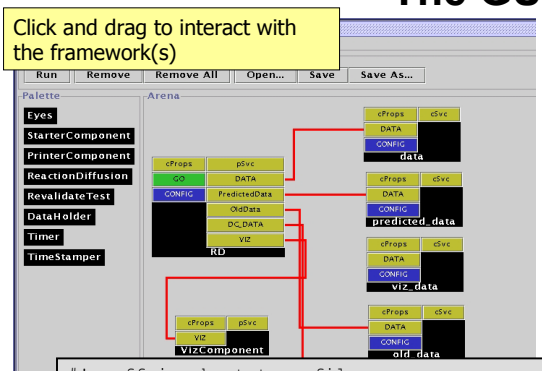


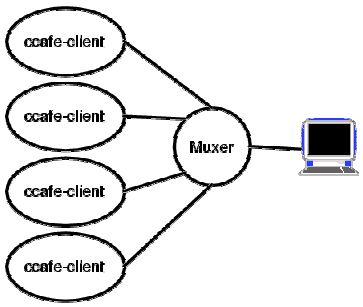
The Ccaffeine GUI

- Java front end to one (or more) *framework* instances running in the background
- Events propagated to all frameworks through a *muxer*
- Framework(s) still need Ccaffeine script to know about available components
- GUI used to instantiate, connect, and configure components (and to launch the whole application as well)
- Usage modes:
  - Compose and launch application from scratch (graphically).
  - Load *pre-composed* applications (the .bld files)

The GUI

Click and drag to interact with the framework(s)





Component paths and types needed by the framework(s) (the .rc files)

```
#!ccaffeine bootstrap file.
# ------- don't change anything ABOVE this line.-------------
path set  /home/elwasif/CCA/tutorial/src/sidl/random-component-c++
path append /home/elwasif/CCA/tutorial/src/sidl/function-component-c++
path append /home/elwasif/CCA/tutorial/src/sidl/integrator-component-c++
path append /home/elwasif/CCA/tutorial/src/sidl/driver-component-c++
repository get randomgen.RandRandomGenerator
repository get functions.LinearFunction
repository get functions.PiFunction
repository get functions.NonlinearFunction
```

SIDL_DLL_PATH environment variable also used for locating component shared libraries!

CCA Common Component Architecture

The Command Line Way: Using Ccaffeine Scripting

- Simple scripting "language" to talk to the framework.
- For the full list of commands:

```
UNIX>ccafe-single
   cca> help
```

- Some commands:
  - `path set  <initial path to components>`
  - `path append <directory containing component code>`
  - `repository get <component class>`
  - `instaniate <component class> <component name>`
  - `connect <use component name> <use port name> \`
    `<provide component name> <provide port name>`
  - `go <component name> <Go port name>`
  - `bye`

CCA Common Component Architecture

Quick run-through of the Ccaffeine scripting language

- Scripting language does everything that the GUI does
- Warning: there are two files that Ccaffeine uses to locate and load component libraries:
  - "rc" and script files for building and running apps
  - GUI ".bld" files that store state saved by the Ccaffeine GUI

  These are not the same and will give, sometimes spectacular, undefined behavior when used improperly.

Example: example1_rc

```
#!ccaffeine bootstrap file.
# ------- don't change anything ABOVE this

path set  /home/elwasif/CCA/tutorial/random-component-c++
path append /home/elwasif/CCA/tutorial/function-component-c++
path append /home/elwasif/CCA/tutorial/integrator-component-c++
path append /home/elwasif/CCA/tutorial/driver-component-c++
# load components into the "pallet"

repository get functions.PiFunction
repository get integrators.MonteCarloIntegrator
repository get integrators.MidPointIntegrator
repository get integrators.ParallelIntegrator
repository get randomgen.RandRandomGenerator
repository get tutorial.driver
```

SIDL_DLL_PATH environment variable also used for locating component shared libraries!

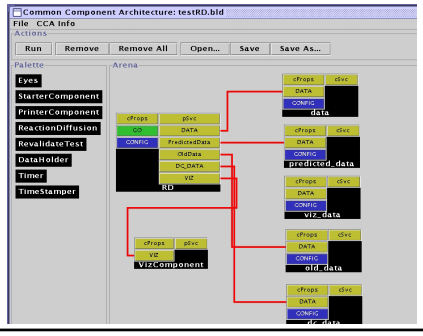
Component classes/types

At this point no components are instantiated, but are simply known to the system

Example (cont.): Instantiation

```
create randomgen.RandRandomGenerator rand
create functions.PiFunction function
create integrators.MonteCarloIntegrator integrator
create tutorial.Driver driver
```

Component instances names



CCA Common Component Architecture

Example (cont.): Connection

```
# Connect uses and provides ports
connect integrator FunctionPort function FunctionPort
connect integrator RandomGeneratorPort rand RandomGeneratorPort
connect driver IntegratorPort integrator IntegratorPort
```


CCA Common Component Architecture

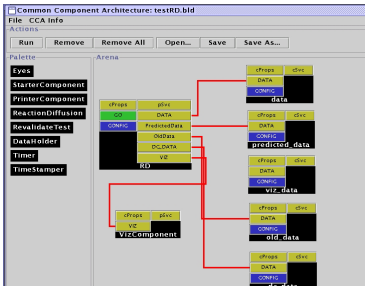
Example (cont.): Application Launch

```
# Good to go()
        go driver GoPort
```

Provided Go port name

At this point Ccaffeine gets completely out of the way

- –So much so that it will not respond until (or if) your application returns from the invocation of the “go()” method
- –There **is** only one thread of control



CCA Common Component Architecture

The third way: Using CCA BuilderService

- Deficiencies of Ccaffeine Scripting
  - Non “standard”
  - No error checking !!!!
- Solution: Use a more “complete” scripting language, e.g. Python
- Why Python? Supported By Babel, popular scripting language
- Strategy:
  - Use a Python driver to assemble the application
  - Talk to the framework through ***`BuilderService`*** interface
  - Still need some Ccaffeine configuration.

CCA Common Component Architecture

The BuilderService Port

- “*Provided*” by the Framework, “*used”* by any component
- Major methods:
  - **`createInstance(instanceName, className, properties)`**
  - **`connect(userID, usePortName, providerID, providPortName)`**
  - See file ***cca.sidl*** for complete interface.
- Many more methods
- Can be “***used***” from any language, Python just more convenient
- See ***`driver-python`*** for details

MonteCarloIntegrator: integrators.depl.cca

- New XML .cca format

Component class name

```
<componentDeployment
  name="example.components.integrators.MonteCarloIntegrator"
  uniqueID="norris@196.128.3.2#9.17.2003.dbg:/MonteCarloIntegrator"
  palletClassAlias="integrators_MonteCarlo">
  <environment>
    <ccaSpec binding="babel"/>
    <library loading="dynamic"
      name="libIntegrator-component-f90.so"
      location="/home/norris/cca/tutorial/src/components/integrators/f90/lib" />
  </environment>
</componentDeployment>
```

Component type: "babel" or "classic"

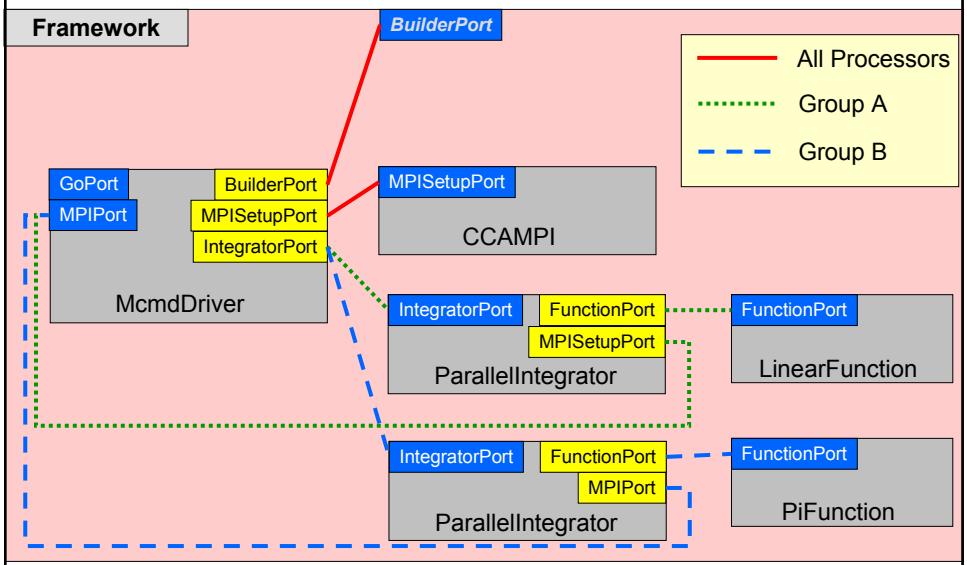
Parallel Components Using MPI

- Single Component Multiple Data (SCMD):
  - Ccaffeine instantiates the same set of components on all processors.
- Multiple Component Multiple Data (MCMD):
  - Ccaffeine instantiates different components on different processors.
  - Needs support components to allow management of the MPI layer.

MCMD using Ccaffeine and MPI

- Need to:
  - Load different components on different processors.
  - Allow the "*driver*" to make the decision based on "global" application state (including MPI state).
  - Avoid re-implementing all MPI calls as component calls (bad for wrapped legacy code).
- Solution:
  - Use ***BuilderServices*** to control component loading.
  - Use "mini MPI" component to allow access to MPI "configuration" calls from various languages (including Python).
  - Structure components to work on a subset of processors (no MPI_COMM_WORLD)

MCMD Port Connections



CCA Common Component Architecture

MCMD Application Logic

- MCMD Driver:
  - Provides ***MPIPort***
  - Instantiate and connect infrastructure components (e. g. ***MPISetupComponent***).
  - Partition MPI_COMM_WORLD based on application logic.
  - Instantiate and connect other components.
- Parallel components
  - Use ***MPIPort*** port to acquire proper communicator (as determined by the driver).
  - Can use **MPI_COMM_WORLD** if no such connection exists.

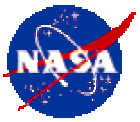
CCA Common Component Architecture

MCMD Issues

- All components share the same MPI library (linked to the framework, or linked to each component as ***shared object.***)
- ***MPIPort*** port provided by the driver can export more application-specific methods.
- Alternate simple solution: Each component exports ***setCommunicator().***

CCA
Common Component Architecture

A Look at More Complex Component-Based Applications

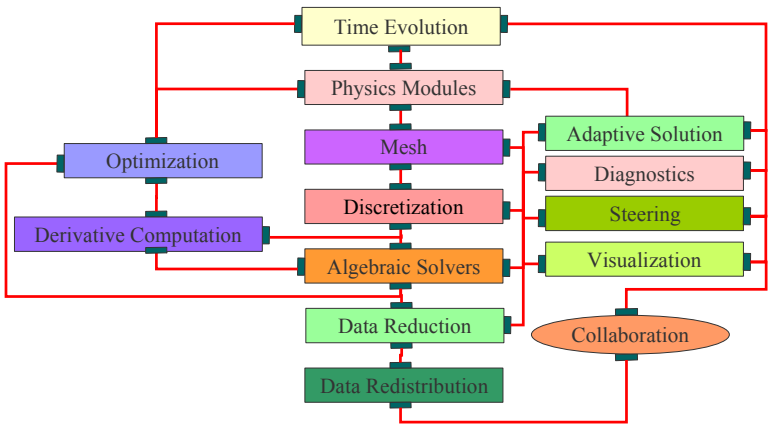


CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org

CCA
Common Component Architecture

Modern Scientific Software Development

- Terascale computing will enable high-fidelity calculations based on multiple coupled physical processes and multiple physical scales
  - Adaptive algorithms and high-order discretization strategies
  - Composite or hybrid solution strategies
  - Sophisticated numerical tools



CCA Common Component Architecture

Overview

- Using components in high performance simulation codes
  - Examples of increasing complexity
  - Performance
    - Single processor
    - Scalability
- Developing components for high performance simulation codes
  - Strategies for thinking about your own application
  - Developing interoperable and interchangeable components

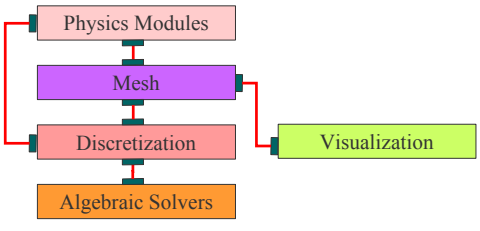
CCA Common Component Architecture

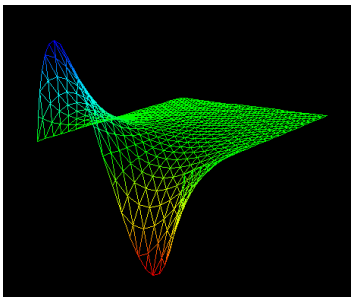
Our Starting Point

$$\nabla^2\varphi\ (x,y) = 0 \in [0,1] \times [0,1]$$

$$\varphi(0,y)=0 \qquad \varphi(1,y)=\sin(2\pi y)$$

$$\delta\varphi/\delta y(x,0) = \delta\varphi/\delta y(x,1) = 0$$





CCA Common Component Architecture

Numerical Solution of Example 1

- Physics: Poisson's equation
- Grid: Unstructured triangular mesh
- Discretization: Finite element method
- Algebraic Solvers: PETSc (Portable Extensible Toolkit for Scientific Computation)
- Visualization: VTK tool
- Original Language: C

CCA Common Component Architecture

Creating Components: Step 1

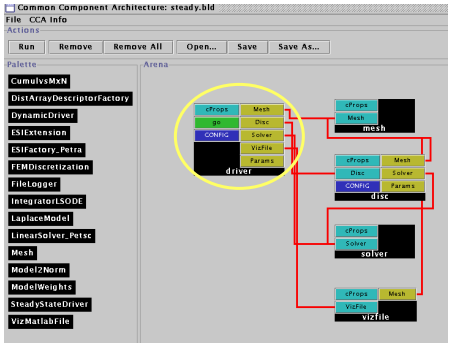
- Separate the application code into well-defined pieces that encapsulate functionalities
  - Decouple code along numerical functionality
    - Mesh, discretization, solver, visualization
    - Physics is kept separate
  - Determine what questions each component can ask of and answer for other components (this determines the ports)
    - Mesh provides geometry and topology (needed by discretization and visualization)
    - Mesh allows user defined data to be attached to its entities (needed by physics and discretization)
    - Mesh *does not* provide access to its data structures
  - If this is not part of the original code design, this is by far the hardest, most time-consuming aspect of componentization

Creating the Components: Step 2

- Writing C++ Components
  - Create an abstract base class for each port
  - Create C++ objects that inherit from the abstract base port class and the CCA component class
  - Wrap the existing code as a C++ object
  - Implement the setServices method
- This process was significantly less time consuming (with an expert present) than the decoupling process
  - Lessons learned
    - Definitely look at an existing, working example for the targeted framework
    - Experts are very handy people to have around ;-)

The Componentized Example

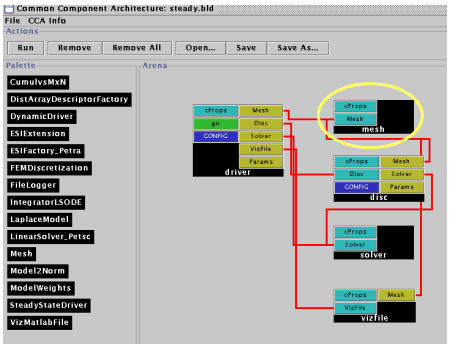
- The Driver Component
  - Responsible for the overall application flow
  - Initializes the mesh, discretization, solver and visualization components
  - Sets the physics parameters and boundary condition information



CCA
Common Component Architecture

The Componentized Example

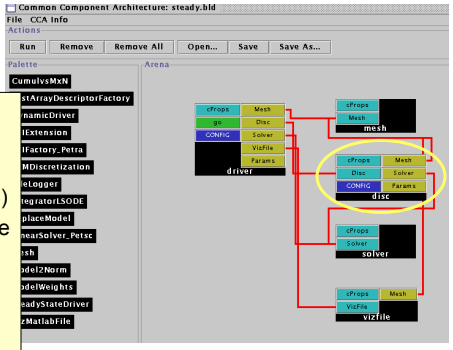
- The Driver Component
- The Mesh Component
  - Provides geometry, topology, and boundary information
  - Provides the ability to attach user defined data as tags to mesh entities
  - Is used by the driver, discretization and visualization components



CCA
Common Component Architecture

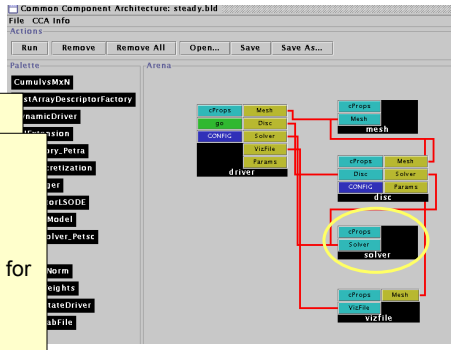
The Componentized Example

- The Driver Component
- The Mesh Component
- The Discretization Component
  - Provides a finite element discretization of basic operators (gradient, Laplacian, scalar terms)
  - Driver determines which terms are included and their coefficients
  - Provides mechanisms for general Dirichlet and Neumann boundary condition matrix manipulations
  - Computes element matrices and assembles them into the global stiffness matrix via set methods on the solver
  - Gathers and scatters vectors to the mesh (in this case φ)



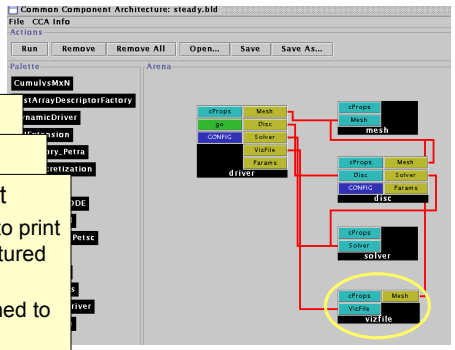
The Componentized Example

- The Driver Component
- The Mesh Component
- The Discretization Component
- The Solver Component
  - Provides access to vector and matrix operations (e.g., create, destroy, get, set)
  - Provides a “solve” functionality for a linear operator



The Componentized Example

- The Driver Component
- The Mesh Component
- The Discretization Component
- The Solver Component
- The Visualization Component
  - Uses the mesh component to print a vtk file of φ on the unstructured triangular mesh
  - Assumes user data is attached to mesh vertex entities



CCA Common Component Architecture

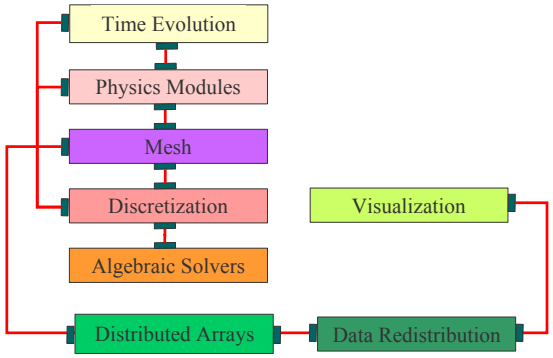
The next step… time dependence

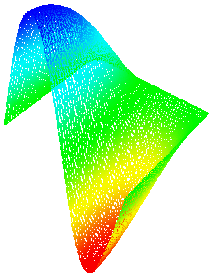
$$\delta\varphi/\delta t = \nabla^2\varphi \ (x,y,t) \in [0,1] \times [0,1]$$

$$\varphi(0,y,t)=0 \qquad \varphi(1,y,t)=.5\sin(2\pi y)\cos(t/2)$$

$$\delta\varphi/\delta y(x,0) = \delta\varphi/\delta y(x,1) = 0$$

$$\varphi(x,y,0)=\sin(.5\pi x)\ \sin\ (2\pi y)$$





CCA Common Component Architecture

Some things change…

- Requires a time integration component
  - Based on the LSODE library (LLNL)
  - Component implementation developed by Ben Allan (SNL)
- Uses a new visualization component
  - Based on AVS
  - Requires an MxN data redistribution component
  - Developed by Jim Kohl (ORNL)
- The MxN redistribution component requires a Distributed Array Descriptor component
  - Similar to HPF arrays
  - Developed by David Bernholdt (ORNL)
- The driver component changes to accommodate the new physics

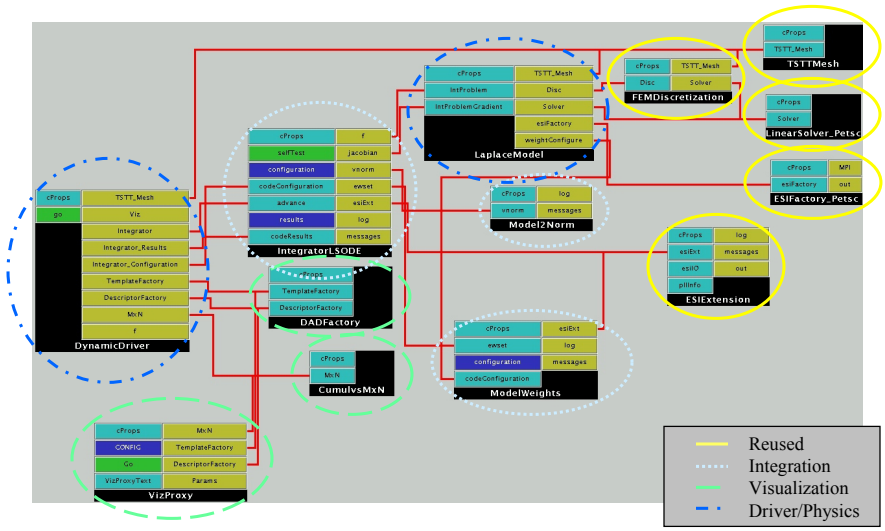
CCA Common Component Architecture

… and some things stay the same

- The mesh component doesn't change
- The discretization component doesn't change
- The solver component doesn't change
  - What we use from the solver component changes
  - Only vectors are needed

CCA Common Component Architecture

The CCA wiring diagram



CCA Common Component Architecture

What did this exercise teach us?

- It was easy to incorporate the functionalities of components developed at other labs and institutions given a well-defined interface and header file.
  - In fact, some components (one uses and one provides) were developed simultaneously across the country from each other after the definition of a header file.
  - Amazingly enough, they usually "just worked" when linked together (and debugged individually).
- In this case, the complexity of the component-based approach was higher than the original code complexity.
  - Partially due to the simplicity of this example
  - Partially due to the limitations of the some of the current implementations of components

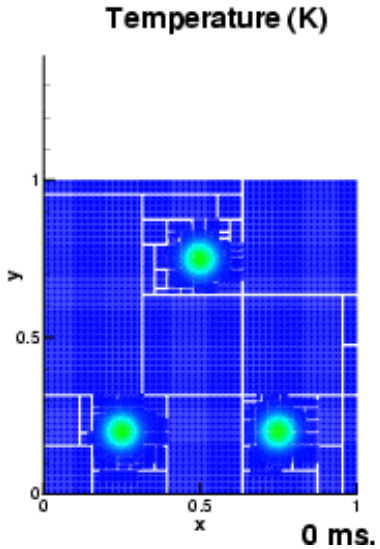
CCA Common Component Architecture

Beyond the heat equation…

- Flame Approximation
  - H_2-Air mixture; ignition via 3 hot-spots
  - 9-species, 19 reactions, stiff chemistry
- Governing equation

$$\frac{\partial Y_i}{\partial t} = \nabla . \alpha \nabla Y_i + \dot{w}_i$$

- Domain
  - 1cm X 1cm domain
  - 100x100 coarse mesh
  - finest mesh = 12.5 micron.
- Timescales
  - O(10ns) to O(10 microseconds)



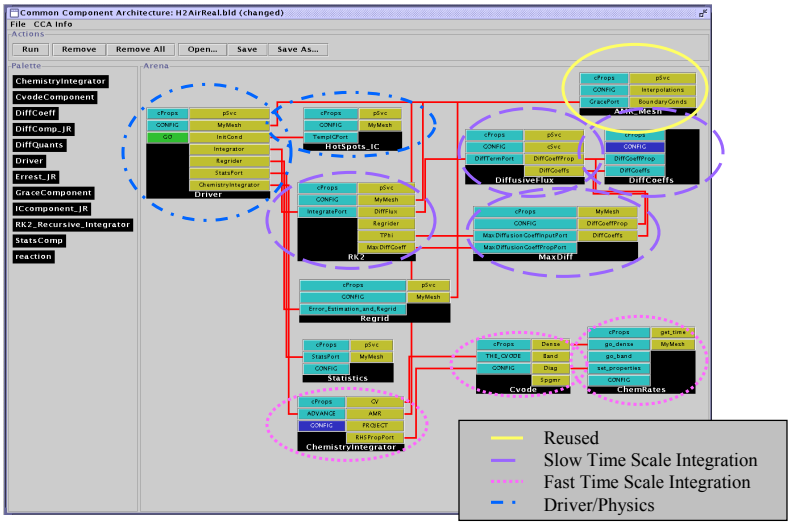
CCA
Common Component Architecture

Numerical Solution

- Adaptive Mesh Refinement: GrACE
- Stiff integrator: CVODE (LLNL)
- Diffusive integrator: 2nd Order Runge Kutta
- Chemical Rates: legacy f77 code (SNL)
- Diffusion Coefficients: legacy f77 code (SNL)
- New code less than 10%

CCA
Common Component Architecture

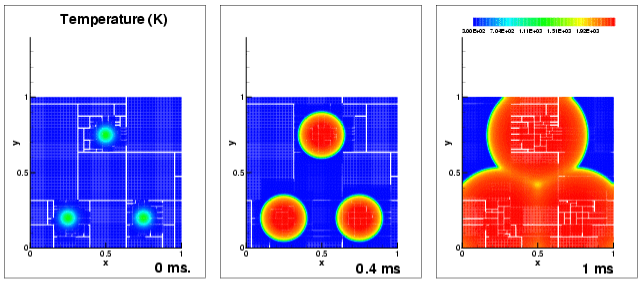
The CCA Wiring Diagram



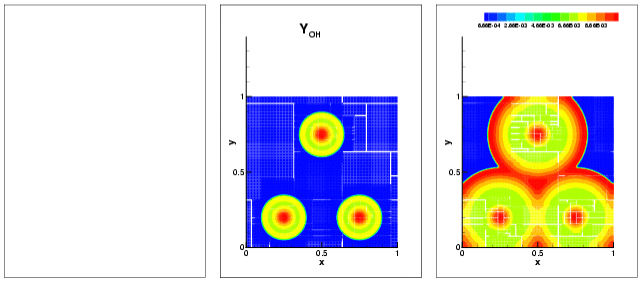
CCA Common Component Architecture

Evolution of the Solution

Temperature



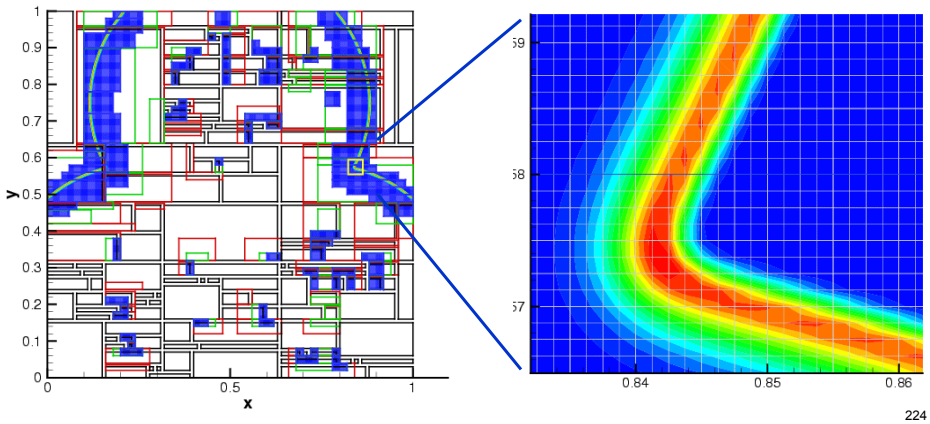
OH Profile



CCA Common Component Architecture

The need for AMR

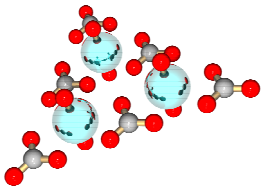
- H_2O_2 chemical subspecies profile
  - Only 100 microns thick (about 10 fine level cells)
  - Not resolvable on coarsest mesh



CCA Common Component Architecture

Computational Chemistry: Molecular Optimization

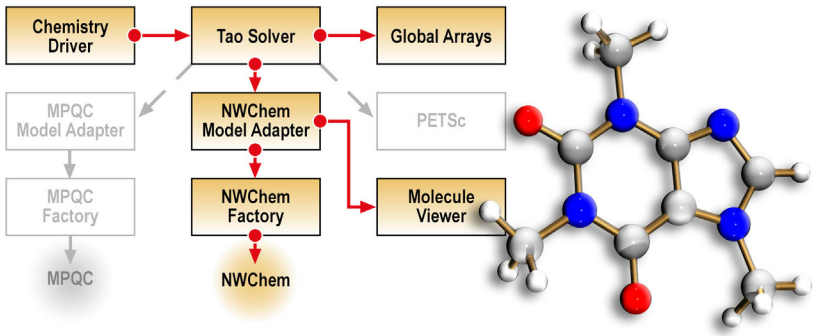
- **Investigators:** Yuri Alexeev (PNNL), Steve Benson (ANL), Curtis Janssen (SNL), Joe Kenny (SNL), Manoj Krishnan (PNNL), Lois McInnes (ANL), Jarek Nieplocha (PNNL), Jason Sarich (ANL), Theresa Windus (PNNL)
- **Goals:** Demonstrate interoperability among software packages, develop experience with large existing code bases, seed interest in chemistry domain
- **Problem Domain:** Optimization of molecular structures using quantum chemical methods



CCA Common Component Architecture

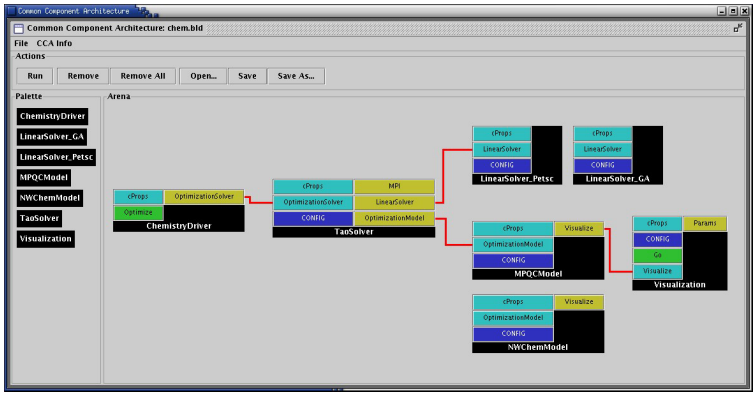
Molecular Optimization Overview

- Decouple geometry optimization from electronic structure
- Demonstrate interoperability of electronic structure components
- Build towards more challenging optimization problems, e.g., protein/ligand binding studies



Components in gray can be swapped in to create new applications with different capabilities.

Wiring Diagram for Molecular Optimization



- Electronic structures components:
  - MPQC (SNL)
    http://aros.ca.sandia.gov/~cljanss/mpqc
  - NWChem (PNNL)
    http://www.emsl.pnl.gov/pub/docs/nwchem
- Optimization components: TAO (ANL)
  http://www.mcs.anl.gov/tao
- Linear algebra components:
  - Global Arrays (PNNL)
    http://www.emsl.pnl.gov:2080/docs/global/ga.html
  - PETSc (ANL)
    http://www.mcs.anl.gov/petsc

Molecular Optimization Summary

- CCA Impact
  - Demonstrated unprecedented interoperability in a domain not known for it
  - Demonstrated value of collaboration through components
  - Gained experience with several very different styles of “legacy” code
- Future Plans
  - Extend to more complex optimization problems
  - Extend to deeper levels of interoperability

CCA Common Component Architecture

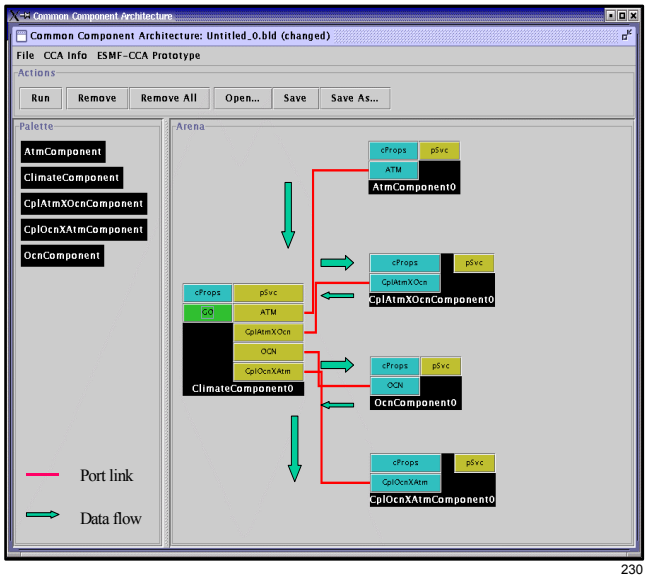
Componentized Climate Simulations

- NASA's ESMF project has a component-based design for Earth system simulations
  - ESMF components can be assembled and run in CCA compliant frameworks such as Ccaffeine.
- Zhou et al (NASA Goddard) has integrated a simple coupled Atmosphere-Ocean model into Ccaffeine and is working on the Cane-Zebiak model, well-known for predicting *El Nino* events.
- Different PDEs for ocean and atmosphere, different grids and time-stepped at different rates.
  - Synchronization at ocean-atmosphere interface; essentially, interpolations between meshes
  - Ocean & atmosphere advanced in sequence
- Intuitively : Ocean, Atmosphere and 2 coupler components
  - 2 couplers : atm-ocean coupler and ocean-atm coupler.
  - Also a Driver / orchestrator component.

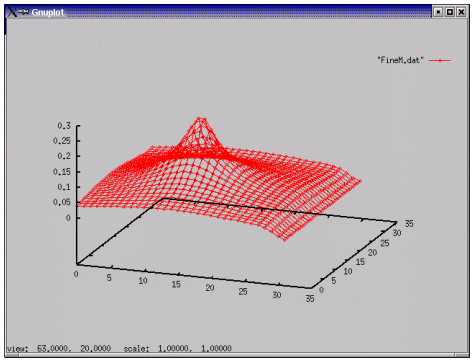
CCA Common Component Architecture

Coupled Atmosphere-Ocean Model Assembly

- **Climate Component :**
  - Schedule component coupling
- **Data flow is via pointer NOT data copy.**
  - All components in C++; run in CCAFFEINE.
- **Multiple ocean models with the same interface**
  - Can be selected by a user at runtime

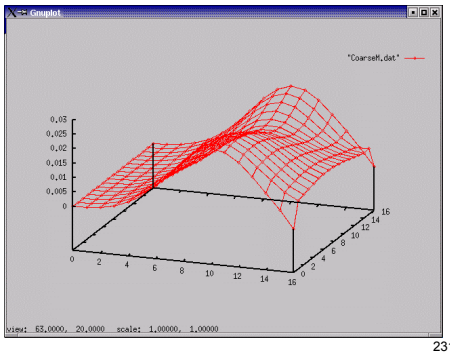


Simulation Results



A non-uniform ocean field variable (e.g., current)

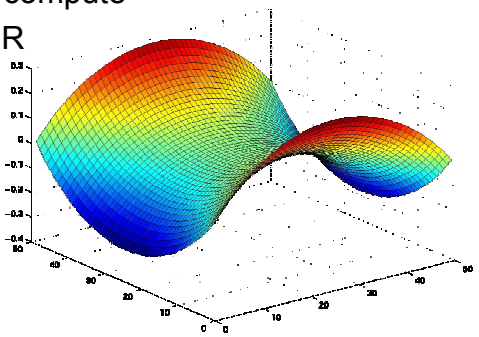
…changes a field variable (e.g.,wind) in the atmosphere !



Unconstrained Minimization Problem

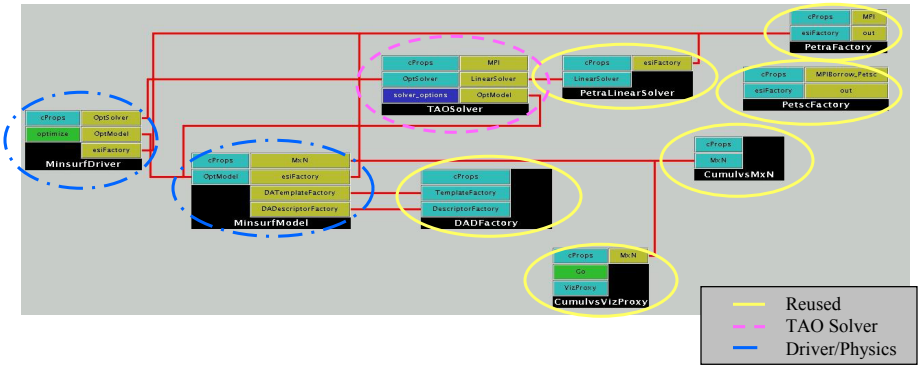
- Given a rectangular 2-dimensional domain and boundary values along the edges of the domain
- Find the surface with minimal area that satisfies the boundary conditions, i.e., compute

  min f(x), where f: R $\rightarrow$ R
- Solve using optimization components based on TAO (ANL)



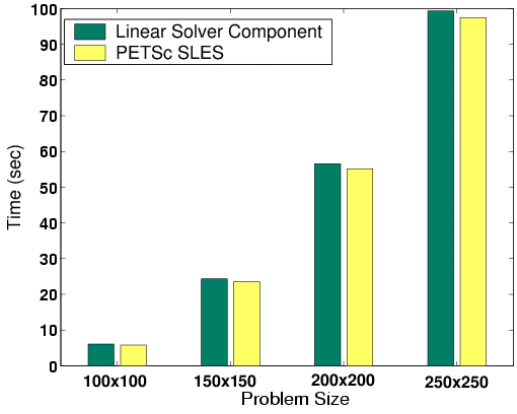
CCA Common Component Architecture

Unconstrained Minimization Using a Structured Mesh



CCA Common Component Architecture

Component Overhead



Aggregate time for linear solver component in unconstrained minimization problem.

- Negligible overhead for component implementation and abstract interfaces when using appropriate levels of abstraction
- Linear solver component currently supports any methods available via the ESI interfaces to PETSc and Trilinos; plan to support additional interfaces the future, e.g., those under development within the TOPS center
- Here: Use the conjugate gradient method with no-fill incomplete factorization preconditioning

CCA Common Component Architecture

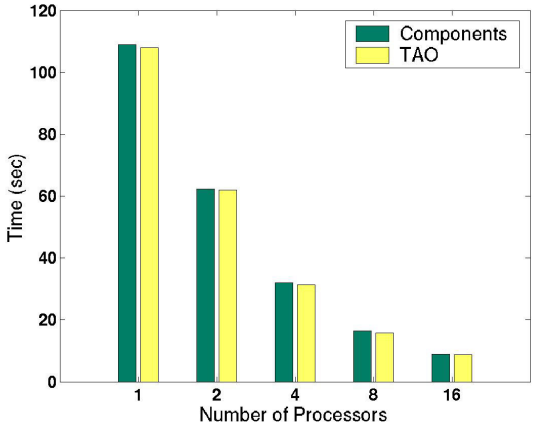
Overhead from Component Invocation

- Invoke a component with different arguments
  - Array
  - Complex
  - Double Complex
- Compare with f77 method invocation
- Environment
  - 500 MHz Pentium III
  - Linux 2.4.18
  - GCC 2.95.4-15
- Components took 3X longer
- Ensure granularity is appropriate!
- Paper by Bernholdt, Elwasif, Kohl and Epperly

Function arg type	f77	Component
Array	80 ns	224ns
Complex	75ns	209ns
Double complex	86ns	241ns

CCA Common Component Architecture

Scalability on a Linux Cluster



Total execution time for the minimum surface minimization problem using a fixed-sized 250x250 mesh.

- Newton method with line search
- Solve linear systems with the conjugate gradient method and block Jacobi preconditioning (with no-fill incomplete factorization as each block's solver, and 1 block per process)
- Negligible component overhead; good scalability

CCA Common Component Architecture

List of Component Re-Use

- Various services in Ccaffeine
- Integrator
  - *IntegratorLSODE (2)*
  - *RK2 (2)*
- Linear solvers
  - *LinearSolver_Petra (4)*
  - *LinearSolver_PETSc (4)*
- AMR
  - *AMRmesh (3)*
- Data description
  - *DADFactory (3)*
- Data redistribution
  - *CumulvsMxN (3)*
- Visualization
  - *CumulvsVizProxy (3)*

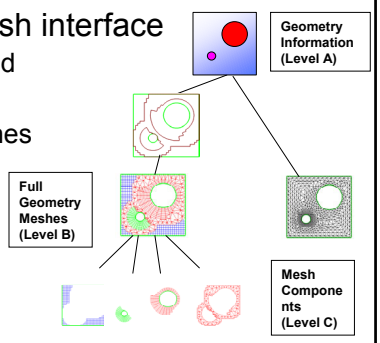
Component interfaces to numerical libraries

Component interfaces to parallel data management and visualization tools

CCA Common Component Architecture

The Next Level

- Common Interface Specification
  - Provides plug-and-play interchangeability
  - Requires domain specific experts
  - Typically a difficult, time-consuming task
  - A success story: MPI
- A case study… the TSTT/CCA mesh interface
  - TSTT = Terascale Simulation Tools and Technologies (www.tstt-scidac.org)
  - A DOE SciDAC ISIC focusing on meshes and discretization
  - Goal is to enable
    - hybrid solution strategies
    - high order discretization
    - Adaptive techniques

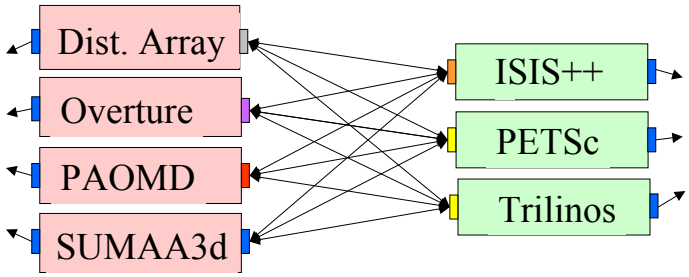


CCA Common Component Architecture

Current Situation

Current Situation

- Public interfaces for numerical libraries are unique
- *Many-to-Many* couplings require *Many*^{2} interfaces
  - Often a heroic effort to understand the inner workings of both codes
  - Not a scalable solution

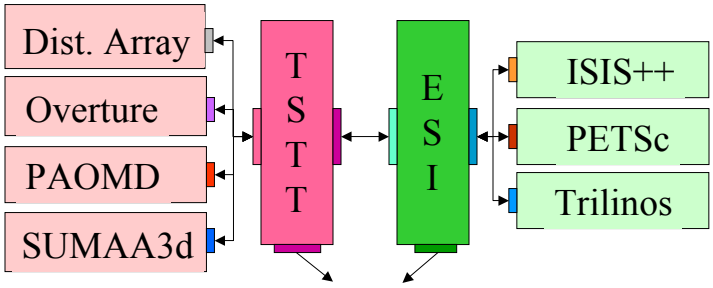


CCA Common Component Architecture

Common Interface Specification

Reduces the *Many-to-Many* problem to a *Many-to-One* problem

- Allows interchangeability and experimentation
- Challenges
  - Interface agreement
  - Functionality limitations
  - Maintaining performance



CCA Common Component Architecture

TSTT Philosophy

- Create a small set of interfaces that existing packages can support
  - AOMD, CUBIT, Overture, GrACE, …
  - Enable both interchangeability and interoperability
- Balance performance and flexibility
- Work with a large tool provider and application community to ensure applicability
  - Tool providers: TSTT and CCA SciDAC centers
  - Application community: SciDAC and other DOE applications

CCA Common Component Architecture

Basic Interface

- Enumerated types
  - Entity Type: VERTEX, EDGE, FACE, REGION
  - Entity Topology: POINT, LINE, POLYGON, TRIANGLE, QUADRILATERAL, POLYHEDRON, TETRAHEDRON, HEXAHEDRON, PRISM, PYRAMID, SEPTAHEDRON
- Opaque Types
  - Mesh, Entity, Workset, Tag
- Required interfaces
  - Entity queries (geometry, adjacencies), Entity iterators, Array-based query, Workset iterators, Mesh/Entity Tags, Mesh Services

CCA Common Component Architecture

Issues that have arisen

- Nomenclature is harder than we first thought
- Cannot achieve the 100 percent solution, so...
  - What level of functionality should be supported?
    - Minimal interfaces only?
    - Interfaces for convenience and performance?
  - What about support of existing packages?
    - Are there atomic operations that all support?
    - What additional functionalities from existing packages should be required?
  - What about additional functionalities such as locking?
- Language interoperability is a problem
  - Most TSTT tools are in C++, most target applications are in Fortran
  - How can we avoid the “least common denominator” solution?
  - Exploring the SIDL/Babel language interoperability tool

CCA Common Component Architecture

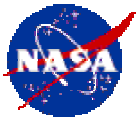
Summary

- Complex applications that use components are possible
  - Combustion
  - Chemistry applications
  - Optimization problems
  - Climate simulations
- Component reuse is significant
  - Adaptive Meshes
  - Linear Solvers (PETSc, Trilinos)
  - Distributed Arrays and MxN Redistribution
  - Time Integrators
  - Visualization
- Examples shown here leverage and extend parallel software and interfaces developed at different institutions
  - Including CUMULVS, ESI, GrACE, LSODE, MPICH, PAWS, PETSc, PVM, TAO, Trilinos, TSTT.
- Performance is not significantly affected by component use
- Definition of domain-specific common interfaces is key

Componentizing your own application

- The key step: think about the decomposition strategy
  - By physics module?
  - Along numerical solver functionality?
  - Are there tools that already exist for certain pieces? (solvers, integrators, meshes?)
  - Are there common interfaces that already exist for certain pieces?
  - Be mindful of the level of granularity
- Decouple the application into pieces
  - Can be a painful, time-consuming process
- Incorporate CCA-compliance
- Compose your new component application
- Enjoy!

CCA Status and Plans



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

CCA Common Component Architecture

CCTTSS Research Thrust Areas and Main Working Groups

- Scientific Components
  - Scientific Data Objects

  Lois Curfman McInnes, ANL (curfman@mcs.anl.gov)
- "MxN" Parallel Data Redistribution

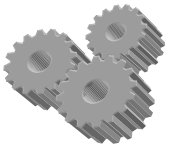
  Jim Kohl, ORNL (kohlja@ornl.gov)
- Frameworks
  - Language Interoperability / Babel / SIDL
  - Component Deployment / Repository

  Gary Kumfert, LLNL (kumfert@llnl.gov)
- User Outreach

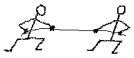
  David Bernholdt, ORNL (bernholdtde@ornl.gov)

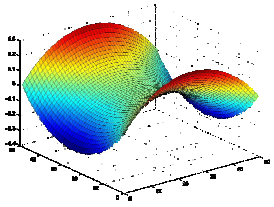
CCA Common Component Architecture

Scientific Components



- Abstract Interfaces and Component Implementations
  - Mesh management
  - Linear, nonlinear, and optimization solvers
  - Multi-threading and load redistribution
  - Visualization and computational steering
- Quality of Service Research
- Fault Tolerance
  - Components and Frameworks



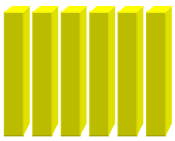


CCA Common Component Architecture

Scientific Components Extended R&D Agenda

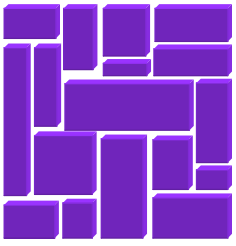
- Complete development of abstract interfaces and base component prototypes
- Advanced component development
  - Second-level component extensions
  - Application-specific components for chemistry and climate
- Implement fault tolerance and recovery mechanisms
- Develop quality of service models for numerical components
  - Integrate QoS system into repository
- Develop interfaces and implementations for multi-level nonlinear solvers and hybrid mesh management schemes
  - Collaboration with TOPS and TSTT centers

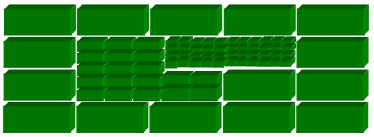
CCA Common Component Architecture



Scientific Data Objects & Interfaces

- Define “Standard” Interfaces for HPC Scientific Data
  - Descriptive, Not (Necessarily) Generative…
- Basic Scientific Data Object
  - David Bernholdt, ORNL
- Structured & Unstructured Mesh
  - Lori Freitag, LLNL
  - Collaboration with SciDAC TSTT Center
- Block Structured AMR
  - Phil Colella, LBNL
  - Collaboration with APDEC & TSTT





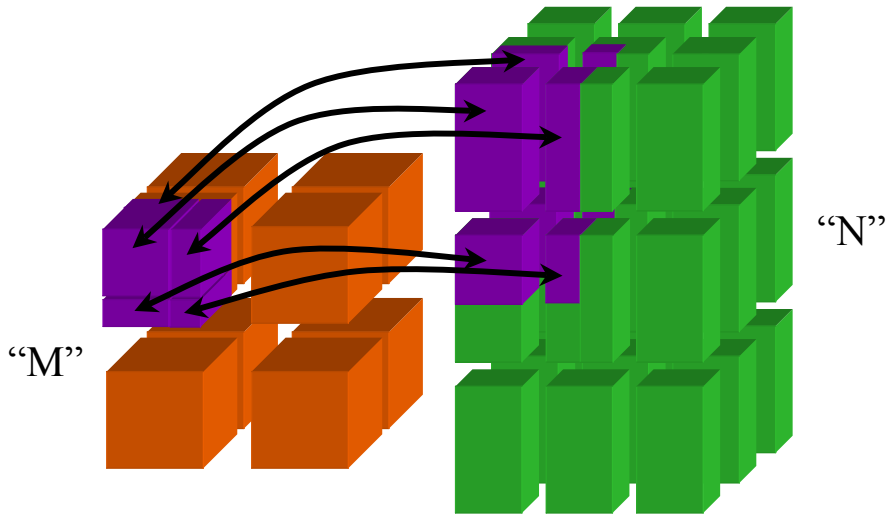
CCA Common Component Architecture

Basic Scientific Data Interfaces

- Low Level, Raw Data
  - Supports high performance access to memory
  - Based on IOVec
    (e.g. http://www-sld.slac.stanford.edu/HELP/POSIX/DATA_STRUCTURES/IOVEC)
    - Assumes a contiguous memory block
    - Supports basic data types such as integer, float, double
    - No topology information
- Local & Distributed Arrays
  - Abstract interfaces for higher-level data description
    - 1D, 2D, 3D dense arrays
    - Various distribution strategies
      - HPF-like decomposition types (Block/Cyclic…)

CCA Common Component Architecture

"MxN" Parallel Data Redistribution: The Problem…



CCA Common Component Architecture

"MxN" Parallel Data Redistribution: The Problem…

- Create complex scientific simulations by coupling together multiple parallel component models
  - Share data on "M" processors with data on "N"
    - M != N ~ Distinct Resources (Pronounced "M by N")
  - Model coupling, e.g., climate, solver / optimizer
  - Collecting data for visualization
    - Mx1; increasingly MxN (parallel rendering clusters)
- Define "standard" interface
  - Fundamental operations for any parallel data coupler
    - Full range of synchronization and communication options

CCA Common Component Architecture

Hierarchical MxN Approach

- Basic MxN Parallel Data Exchange
  - Component implementation
  - Initial prototypes based on CUMULVS & PAWS
    - Interface generalizes features of both
- Higher-Level Coupling Functions
  - Time & grid (spatial) interpolation, flux conservation
  - Units conversions…
- "Automatic" MxN Service via Framework
  - Implicit in method invocations, "parallel RMI"

http://www.csm.ornl.gov/cca/mxn/

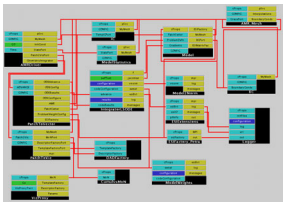
CCA Common Component Architecture

CCA Frameworks

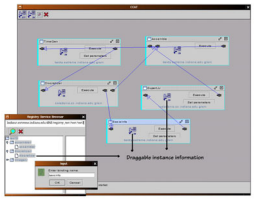
- Component Containers & Run-Time Environments
- Research Areas:
  - Integration of prototype frameworks
    - SCMD/parallel with distributed, bridged for one application
    - Unify framework services & interactions…
  - Language interoperability tools
    - Babel/SIDL, incorporate difficult languages (F90…)
    - Production-scale requirement for application areas
  - Component deployment
    - Component repository, interface lookup & semantics

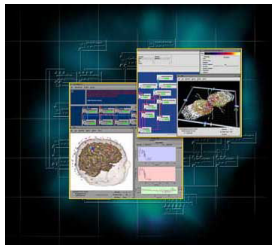
CCA Common Component Architecture

CCA Framework Prototypes



- Ccaffeine
  - SPMD/SCMD parallel
  - Direct connection
- CCAT / XCAT
  - Distributed
  - Network connection
- SCIRun
  - Parallel, multithreaded
  - Direct connection
- Decaf
  - Original language interoperability via Babel…





CCA Common Component Architecture

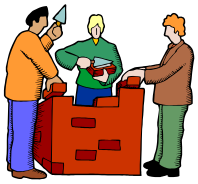
Outreach and Applications Integration

- Tools Not Just “Thrown Over The Fence”…
- Several Outreach Efforts:
  - General education and awareness
    - Tutorials, like this one!
    - Papers, conference presentations
  - Strong liaison with adopting groups
    - Beyond superficial exchanges
    - Real production requirements & feedback
  - Chemistry and climate work within CCTTSS
    - Actual application development work ($$$)
- SciDAC Emphasis
  - More vital ***applied*** advanced computing research!

CCA Common Component Architecture

Active CCA Forum Working Groups

- Adaptive Mesh Refinement
- Generalized Data Objects
- Tutorial Presentations
- Application Domain Groups:
  - Climate, Chemistry
- MxN Data Redistribution
- Embeddable Scripting
- Fortran Users
- Babel Development & Users
- Deployment / XML Schemas
- Ccaffeine Open Framework
- Component-Based Debugging…



See http://www.cca-forum.org/working_groups.html for more info.

CCA Common Component Architecture

Current CCA / CCTTSS Status

- CCA Specification at Version 0.6.2
- Several Operational Prototype Frameworks
- Growing Number of Reusable Component Modules
- Draft specifications for
  - Basic scientific data objects
  - MxN parallel data redistribution
- Demonstration Software ***Available for Download***
  - Several Multi-Component Parallel and Distributed Demonstration Applications
  - Variety of components for: optimization, solvers, meshes, data decompositions, visualization, MxN…
  - RPM packages for easy Linux install!

http://www.cca-forum.org/software.html

CCA Common Component Architecture

CCA Tutorial Summary

- Go Forth and Componentize…
  - And ye shall bear good scientific software
- Come Together for Domain Standards
  - Attain true interoperability & code re-use
- Use The Force:
  - http://www.cca-forum.org/tutorials/
  - *tutorial-wg@cca-forum.org*
  - *cca-forum@cca-forum.org*