CCA
Common Component Architecture

Welcome to the Common Component Architecture Tutorial

ICCS 2005

22 May 2005

Tutorial 3a/3b

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

CCA
Common Component Architecture

Agenda & Table of Contents

CCA
Common Component Architecture

The Common Component Architecture (CCA) Forum

- Combination of standards body and user group for the CCA
- Define Specifications for ***High-Performance*** Scientific Components & Frameworks
- Promote and Facilitate Development of Domain-Specific ***Common 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/

3

CCA
Common Component Architecture

Acknowledgements: Tutorial Working Group

- **People:** Rob Armstrong, David Bernholdt, Randy Bramley, Wael Elwasif, Lori Freitag Diachin, Madhusudhan Govindaraju, Ragib Hasan, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, Sameer Shende, Torsten Wilde, Shujia Zhou
- **Institutions:** ANL, Binghamton U, Indiana U, JPL, LANL, LLNL, NASA/Goddard, ORNL, SNL, U Illinois, U Oregon
- **Computer facilities** provided by the Computer Science Department and University Information Technology Services of Indiana University, supported in part by NSF grants CDA-9601632 and EIA-0202048.

4

CCA
Common Component Architecture

Acknowledgements: The CCA

- **ANL** –Steve Benson, Jay Larson, Ray Loy, Lois Curfman McInnes, Boyana Norris, Everest Ong, Jason Sarich…
- **Binghamton University -** Madhu Govindaraju, Michael Lewis, …
- **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 many more… without whom we wouldn't have much to talk about!

5

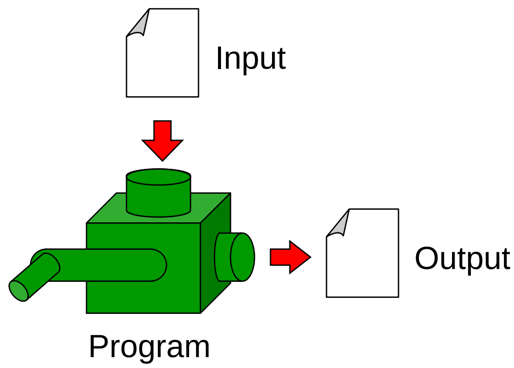
CCA
Common Component Architecture

A Pictorial Introduction to Components in Scientific Computing

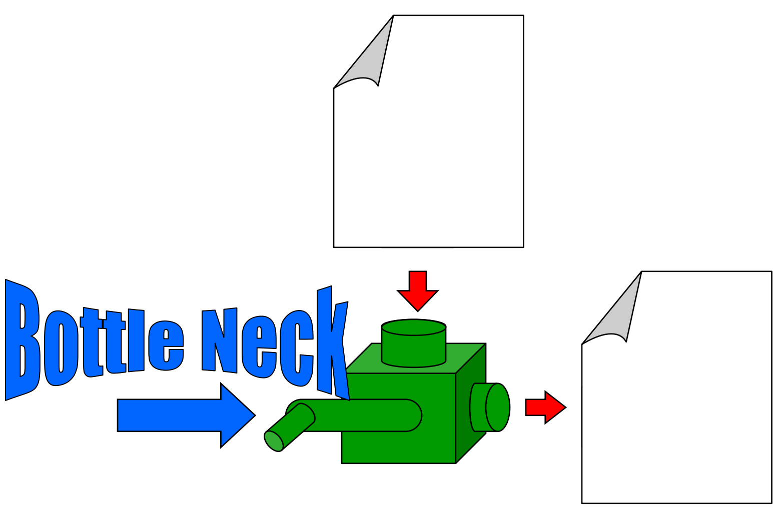
CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org

6

Once upon a time...

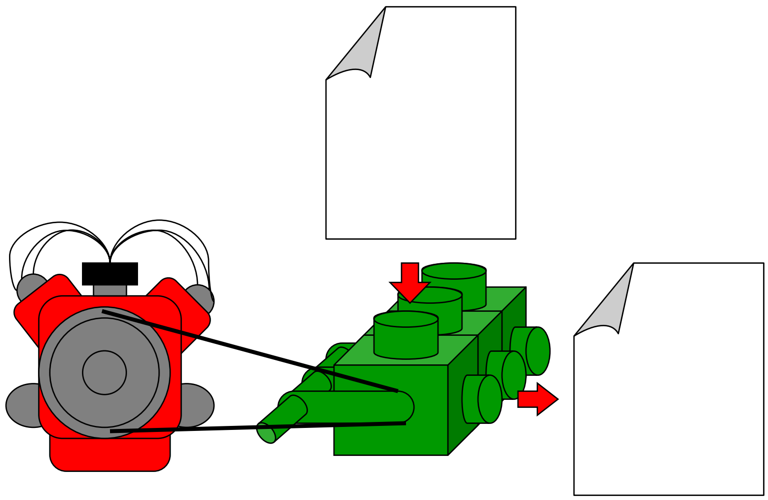


As Scientific Computing grew...



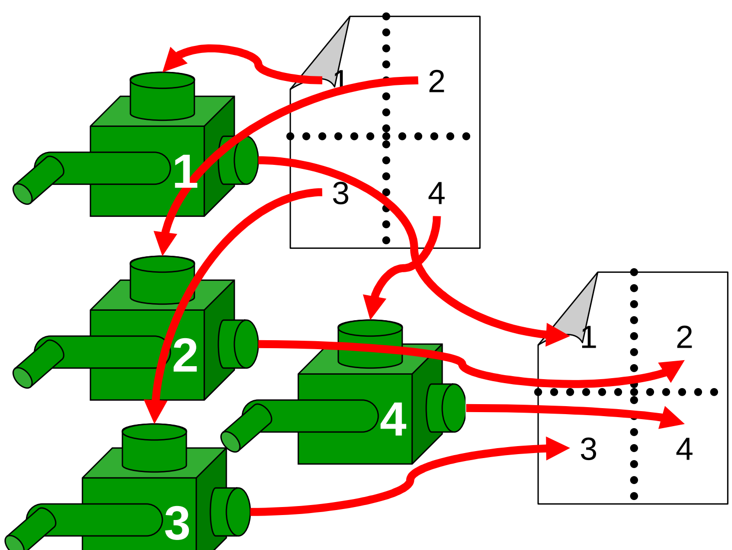
CCA Common Component Architecture

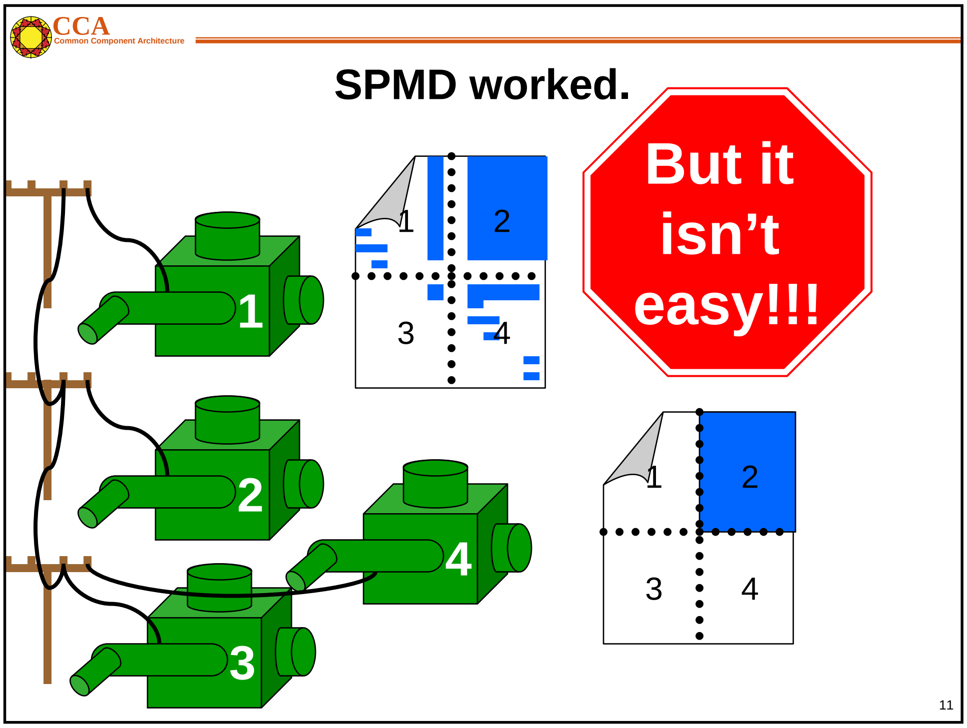
Tried to ease the bottle neck

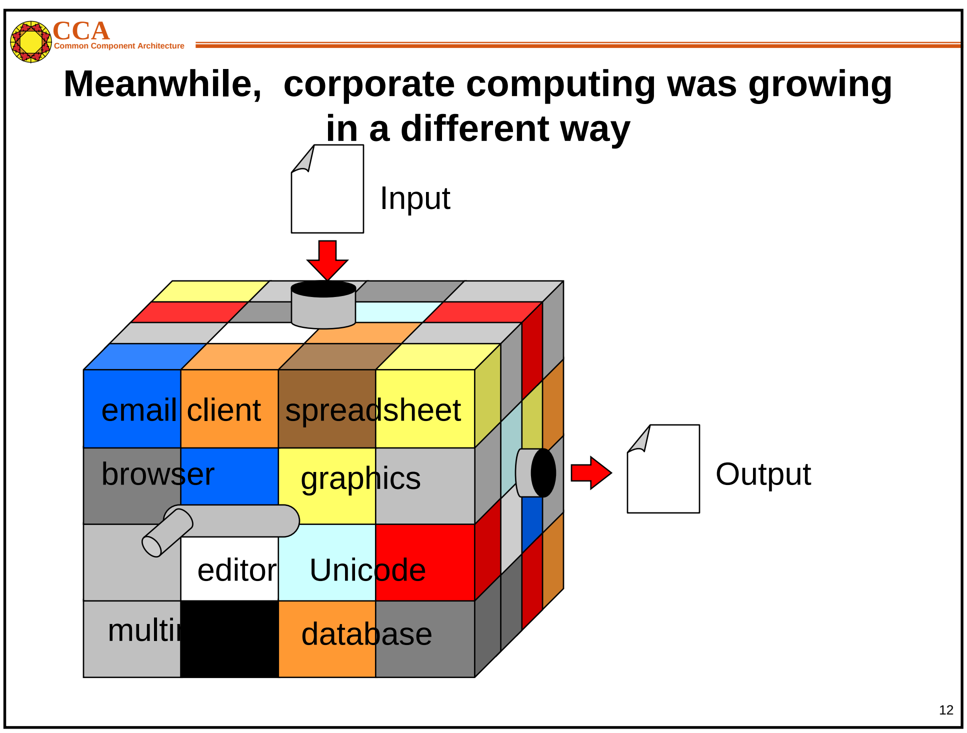


CCA Common Component Architecture

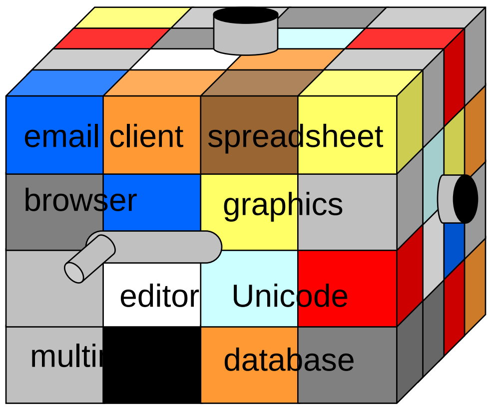
SPMD was born.




CCA
Common Component Architecture
SPMD worked.
But it isn't easy!!!
1
2
3
4
1
2
3
4
1
2
3
4
11

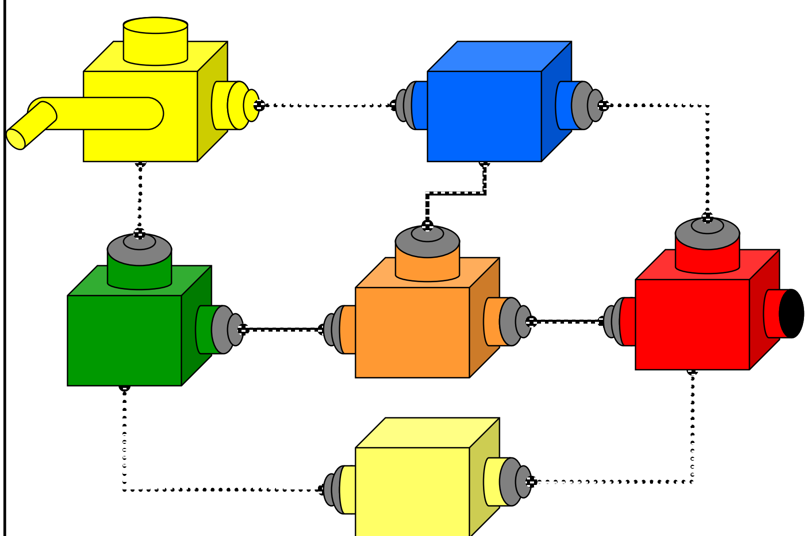

CCA
Common Component Architecture
Meanwhile, corporate computing was growing in a different way
Input
email client spreadsheet
browser graphics
editor Unicode
multi
database
Output
12

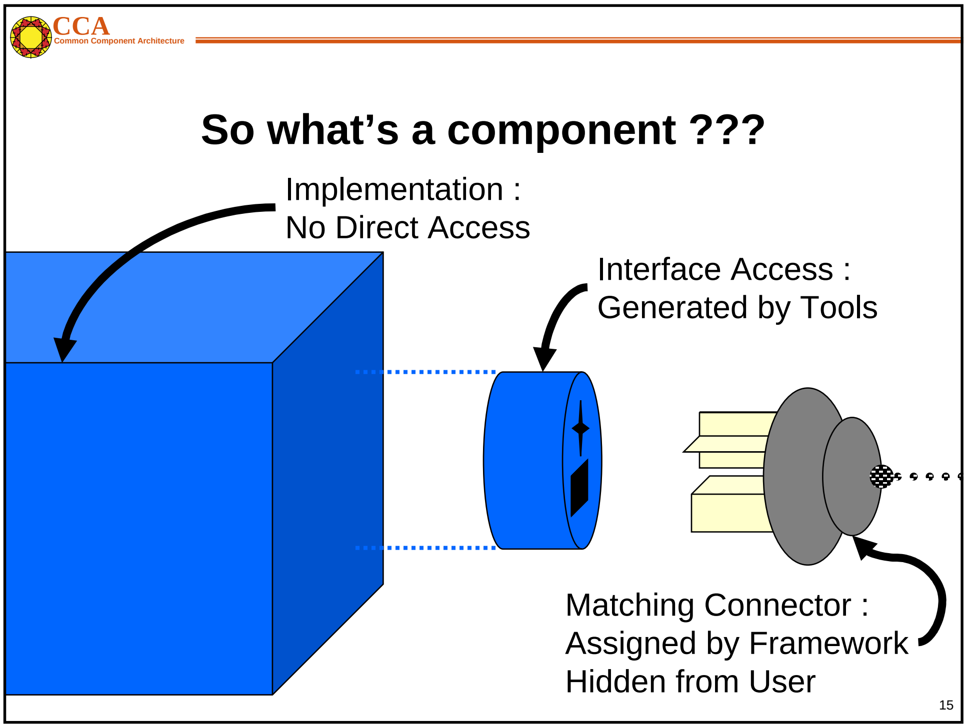
This created a whole new set of problems → complexity

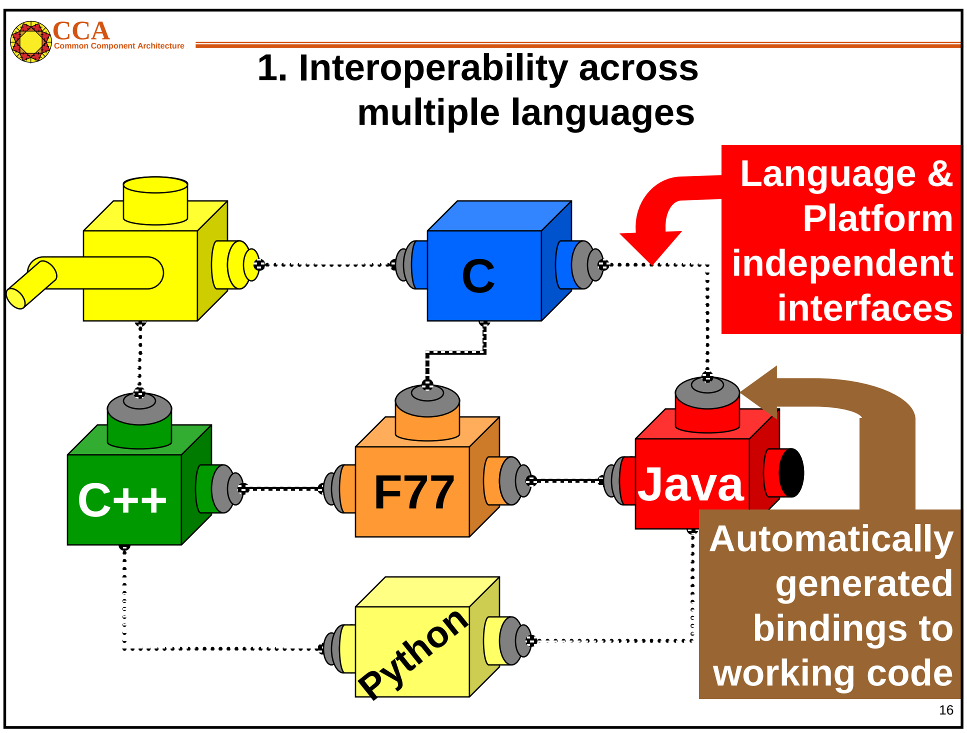


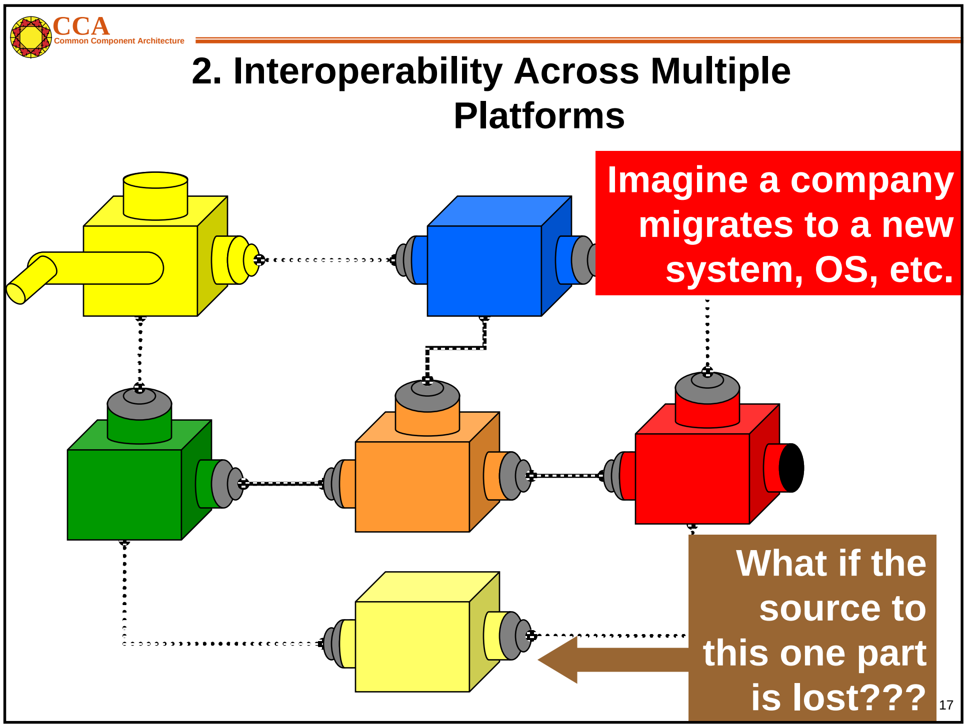
- Interoperability across multiple languages
- Interoperability across multiple platforms
- Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)

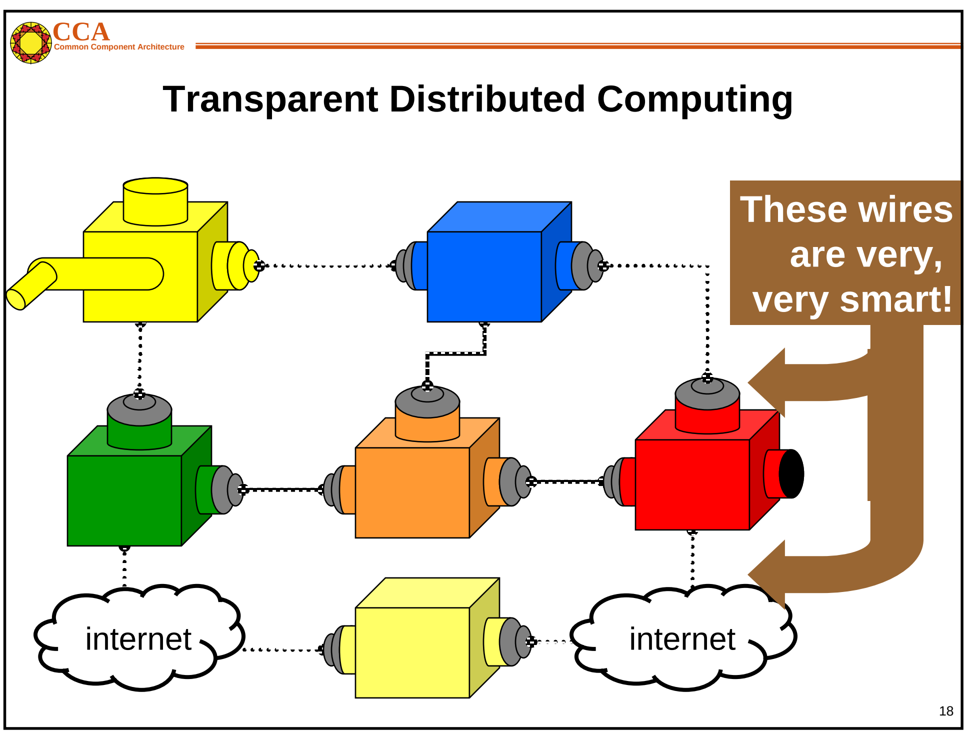
Component Technology addresses these problems



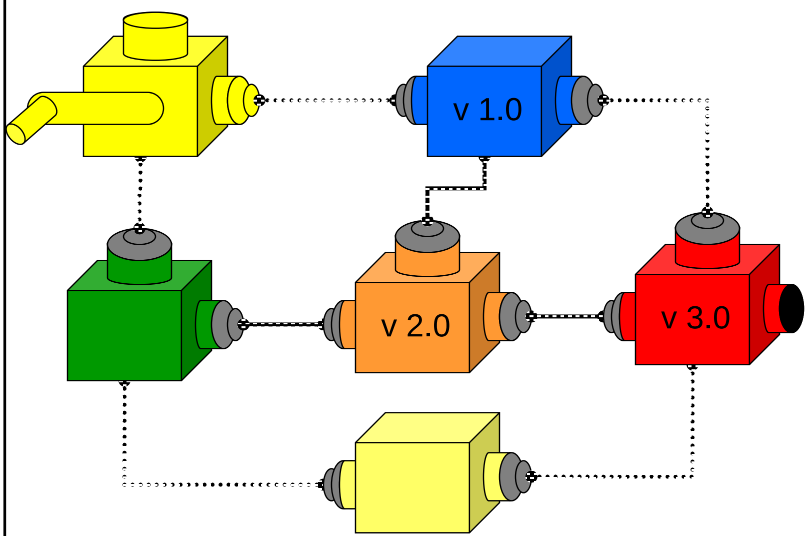

CCA
Common Component Architecture
So what's a component ???
Implementation :
No Direct Access
Interface Access :
Generated by Tools
Matching Connector :
Assigned by Framework
Hidden from User
15


CCA
Common Component Architecture
1. Interoperability across
multiple languages
Language &
Platform
independent
interfaces
C
C++
F77
Java
Python
Automatically
generated
bindings to
working code
16

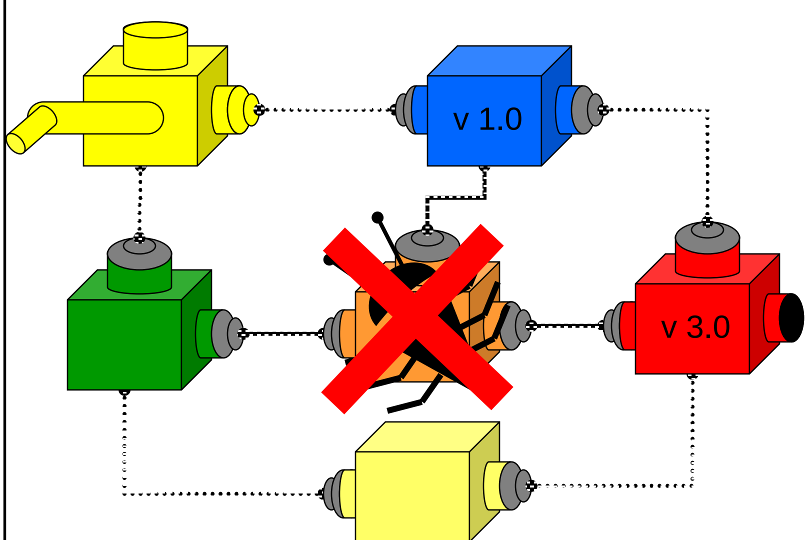

CCA
Common Component Architecture
2. Interoperability Across Multiple Platforms
Imagine a company migrates to a new system, OS, etc.
What if the source to this one part is lost???
17


CCA
Common Component Architecture
Transparent Distributed Computing
These wires are very, very smart!
internet
internet
18

3. Incremental Evolution With Multiple 3rd party software

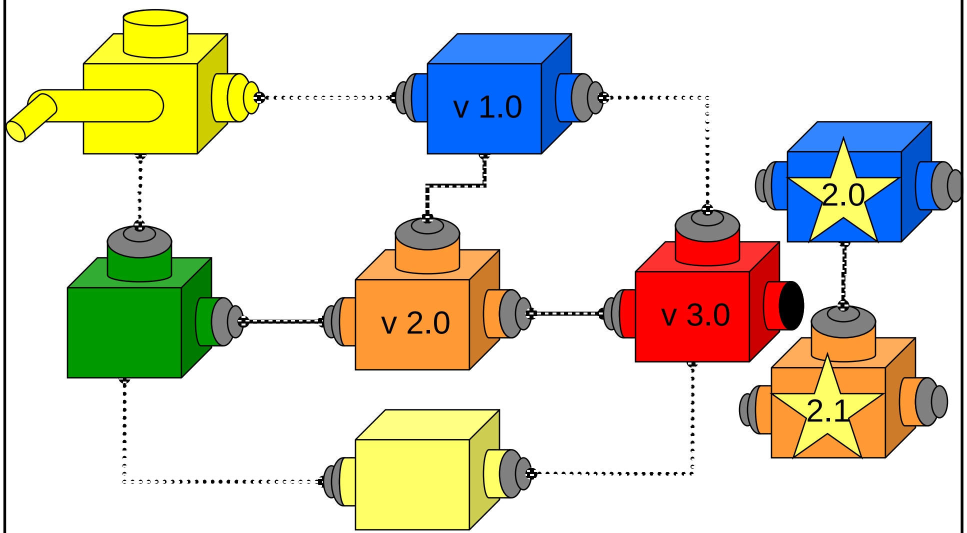


Now suppose you find this bug...



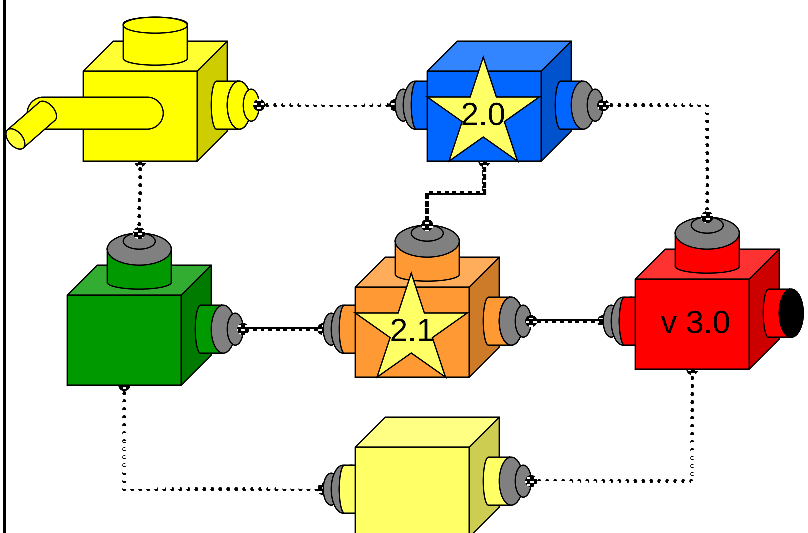
CCA
Common Component Architecture

Good news: an upgrade available
Bad news: there's a dependency

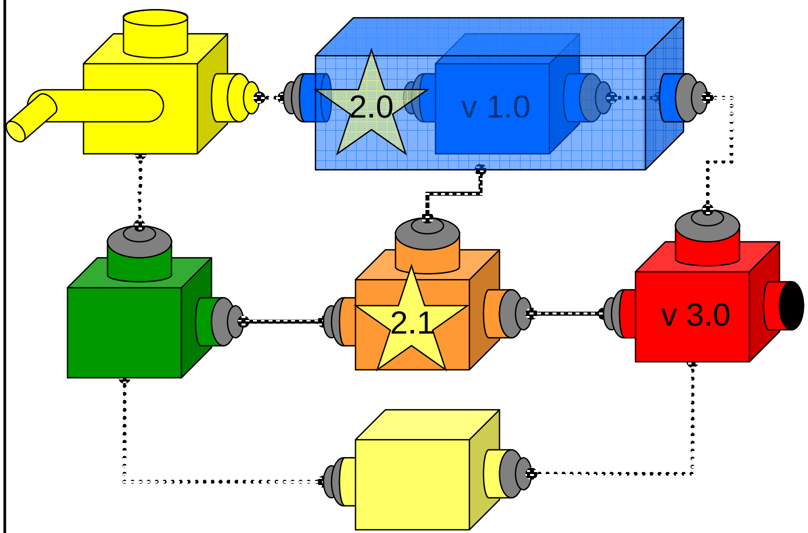


CCA
Common Component Architecture

Great News:
Solvable with Components

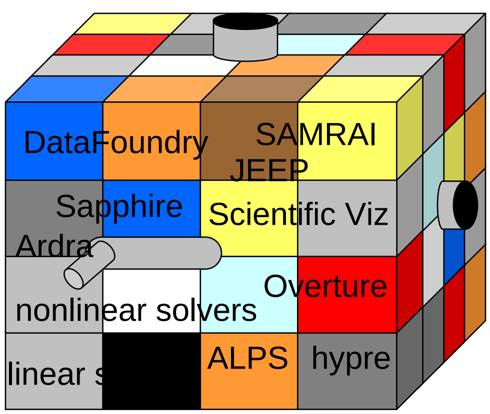


Great News: Solvable with Components



23

Why Components for Scientific Computing → Complexity

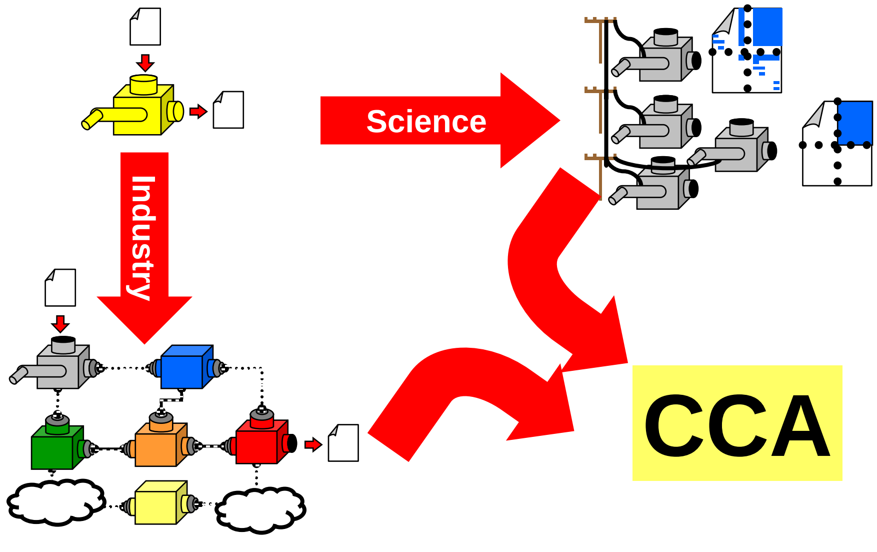


- Interoperability across multiple languages
- Interoperability across multiple platforms
- Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)

24

CCA
Common Component Architecture

The Model for Scientific Component Programming



CCA
Common Component Architecture

An Introduction to Components and the Common Component Architecture

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 and the CCA
- Highlight the special demands of high-performance scientific computing on component environments

27

CCA
Common Component Architecture

Component-Based Software Engineering

- CBSE methodology is an emerging approach to software development
  – Both in research an in practical application
  – Especially popular in business and internet areas
- Addresses software complexity issues
- Increases software productivity

28

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

CCA
Common Component Architecture

What are Components?

- No universally accepted definition in computer science research …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
- Can be composed with other components
  - “Plug and play” model to build applications
  - Composition based on interfaces

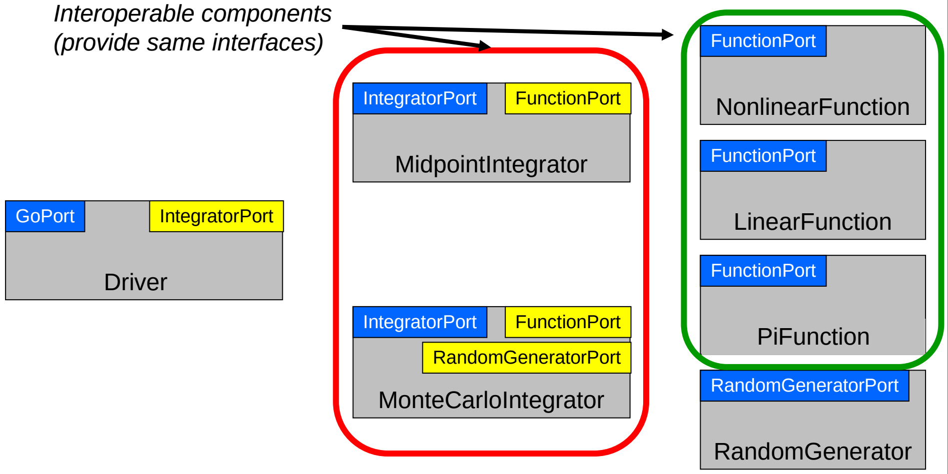
CCA
Common Component Architecture

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

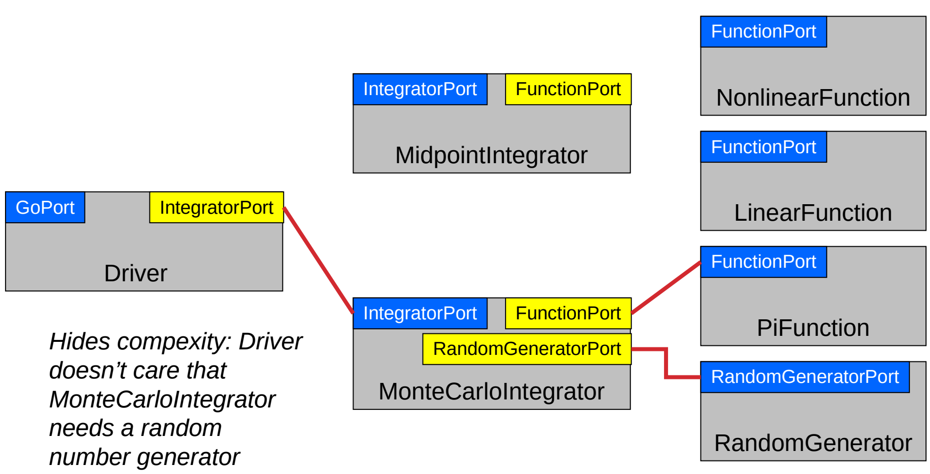
A Simple Example: Numerical Integration Components



33

CCA Common Component Architecture

An Application Built from the Provided Components

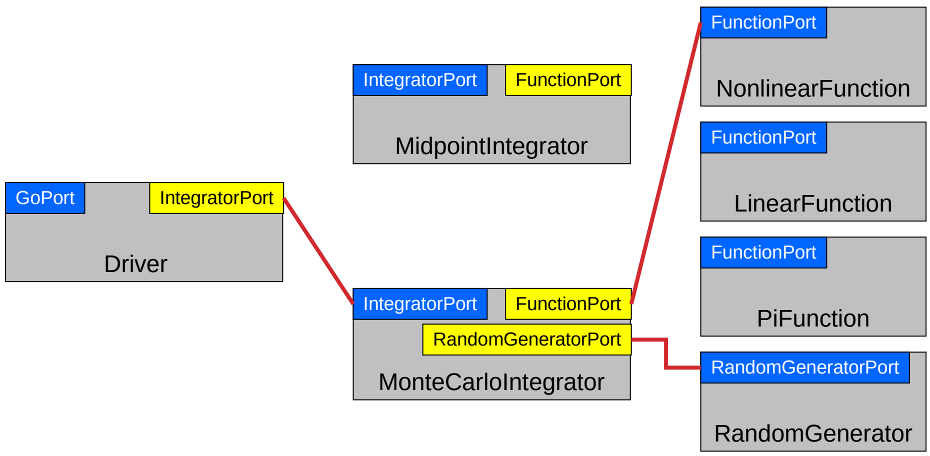


Hides compexity: Driver doesn't care that MonteCarloIntegrator needs a random number generator

34

CCA
Common Component Architecture

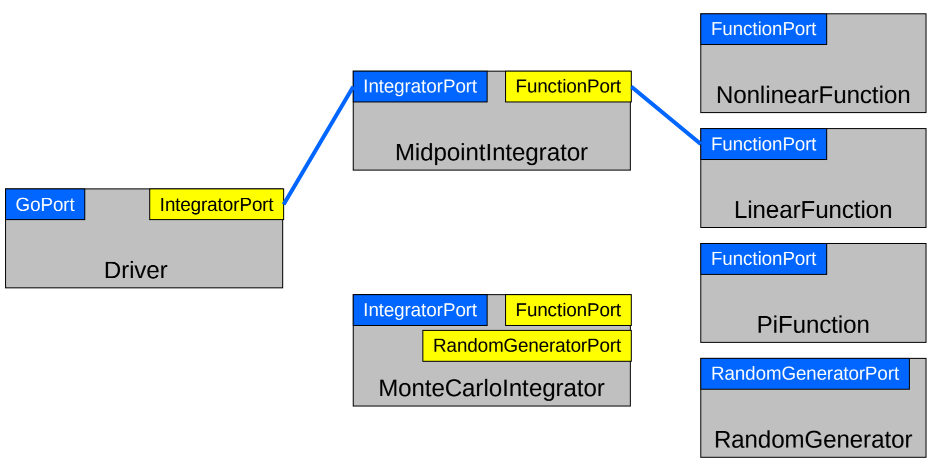
Another Application…



35

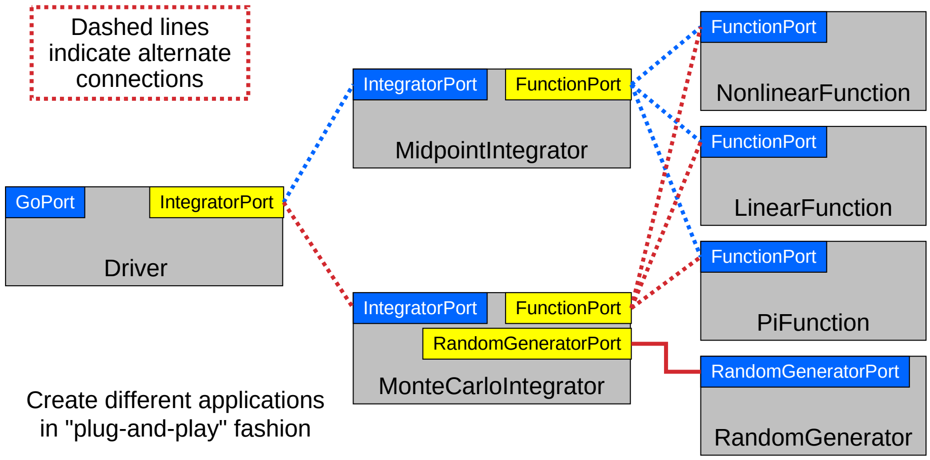
CCA
Common Component Architecture

Application 3…



36

And Many More…



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

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
  - E.g. Cactus, ESMF, PRISM
  - Hypre, Overture, PETSc, POOMA

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
  - e.g. CCA

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…
- Achieving reuse across multiple applications requires agreement on the same interface for all of them

- "Common" or "community" interfaces facilitate reuse and interoperability
  - Typically domain specific
  - Formality of "standards" process varies
  - Significant initial investment for long-term payback
- Biggerstaff's Rule of Threes
  - Must look at at least three systems to understand what is common (reusable)
  - Reusable software requires three times the effort of usable software
  - Payback only after third release

More about community interface development efforts in "Applications" module

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 Component Model (CCM), 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

What is the CCA?

- CCA is a *specification* of a component environment designed for high performance scientific computing
  - Specification is decided by the CCA Forum
    - CCA Forum membership open to all
  - "CCA-compliant" just means conforming to the specification
    - Doesn't require using any of our code!
- 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

43

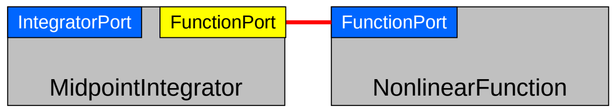
CCA
Common Component Architecture

CCA Philosophy and Objectives

- Local and remote components
  - Support local, HPC parallel, and distributed computing
- High Performance
  - Design should support high-performance mechanisms wherever possible (i.e. minimize copies, extra communications, extra synchronization)
  - Support SPMD and MPMD parallelism
  - Allow user to choose parallel programming models
- Heterogeneity
  - Multiple architectures, languages, run-time systems used simultaneously in an application
- Integration
  - Components should be easy to make and easy to use
- Openness and simplicity
  - CCA spec should be open & usable with open software

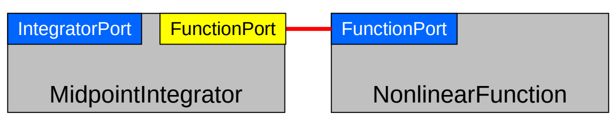
44

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 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

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**

Full details in the hands-on!

CCA Common Component Architecture

Adapting Existing Code into Components

Example in the hands-on!

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

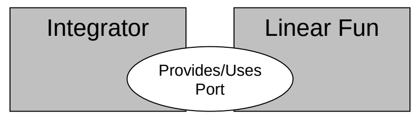
Writing Frameworks

- ***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

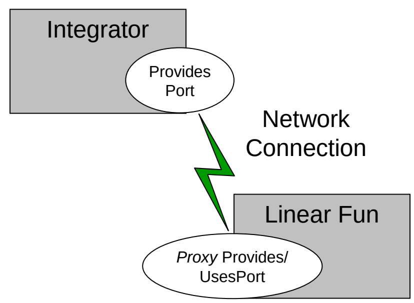
CCA Supports Local, Parallel and Distributed Computing

- "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

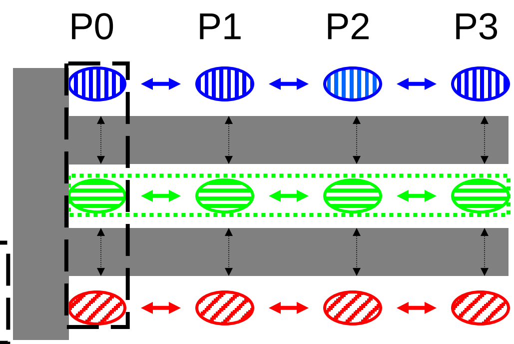
CCA Concepts: "Direct Connection" Maintains Local Performance

- 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

More about performance in the "Applications" module

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

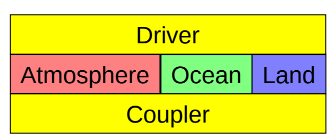
Framework: Gray

MCMD/MPMD also supported

Other component models ignore parallelism entirely

"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

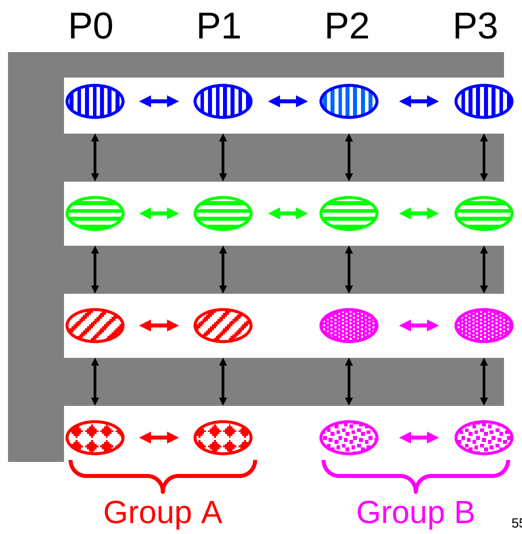


- Approaches
  - Run single parallel framework
    - Driver component that partitions processes and builds rest of application as appropriate (through BuilderService)
  - Run multiple parallel frameworks
    - Link through specialized communications components
    - Link as components (through AbstractFramework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

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

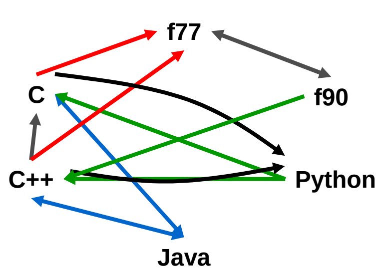


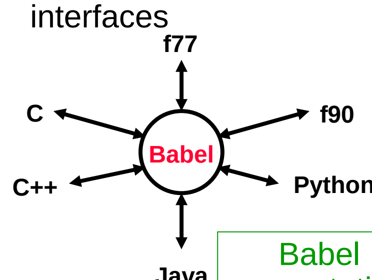
55

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





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

Babel presentation coming up!

56

CCA
Common Component Architecture

Advanced CCA Concepts

- 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
- Meta-component models enable bridging between CCA components and other component(-like) environments
  - e.g. SCIRun Dataflow, Visualization Toolkit (VTk), …

No time to go into detail on these, but ask us for more info after the tutorial

CCA
Common Component Architecture

Component Lifecycle

Additional material in notes

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

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

Steps may be under human or software control

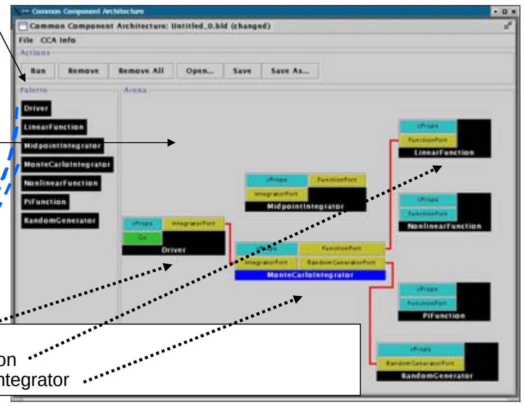
CCA Common Component Architecture

Supplementary material for handouts

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

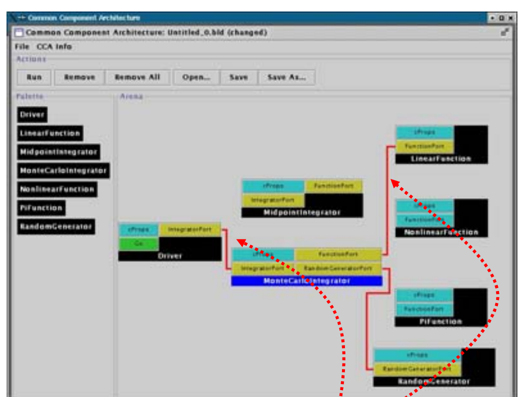
59

CCA Common Component Architecture

Supplementary material for handouts

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

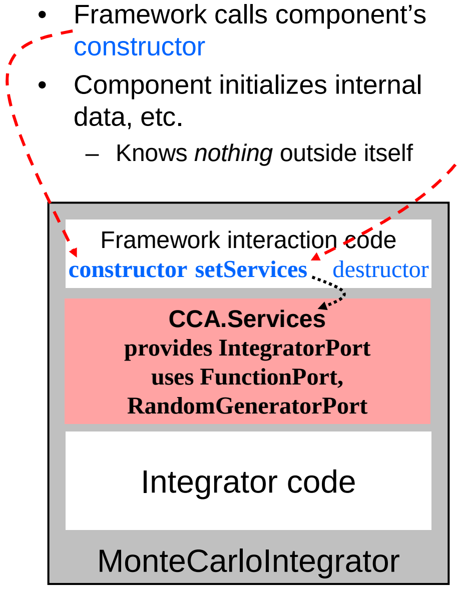


connect	Driver	IntegratorPort	MonteCarloIntegrator	IntegratorPort
connect	MonteCarloIntegrator	FunctionPort	LinearFunction	FunctionPort
…				

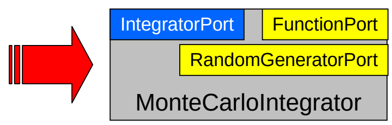
60

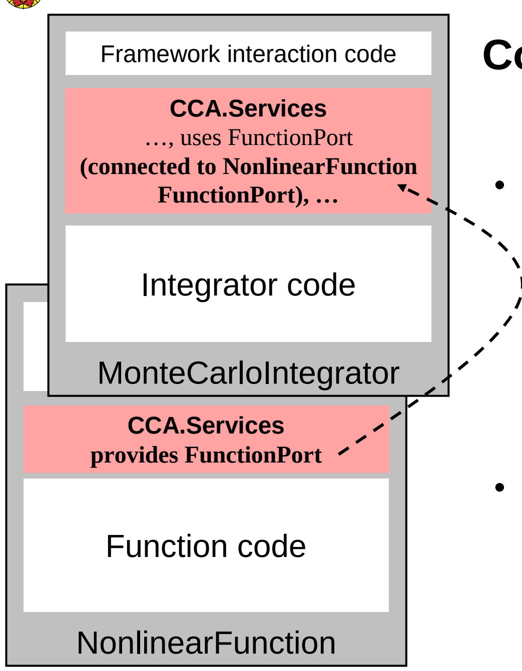
Component's View of Instantiation

- 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





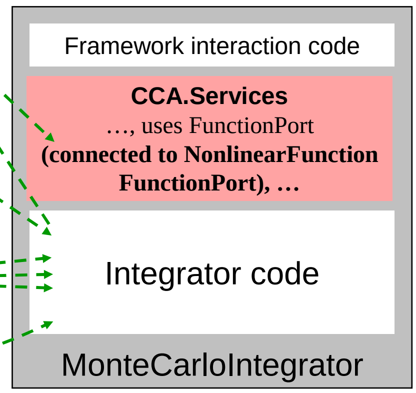
Component's View of Connection

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

Component's View of Using a Port

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

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



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 common interface efforts
- CCA doesn't require (but does support) separation of algorithms/physics from data
  - Generic programming

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 is designed for interoperability
  - Components within a CCA environment
  - CCA environment with other tools, libraries, and frameworks
- 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
- **Language Interoperability**
  - Babel, Scientific Interface Definition Language (SIDL)

CCA
Common Component Architecture

Language Interoperable CCA Components via BABEL

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

CCA
Common Component Architecture

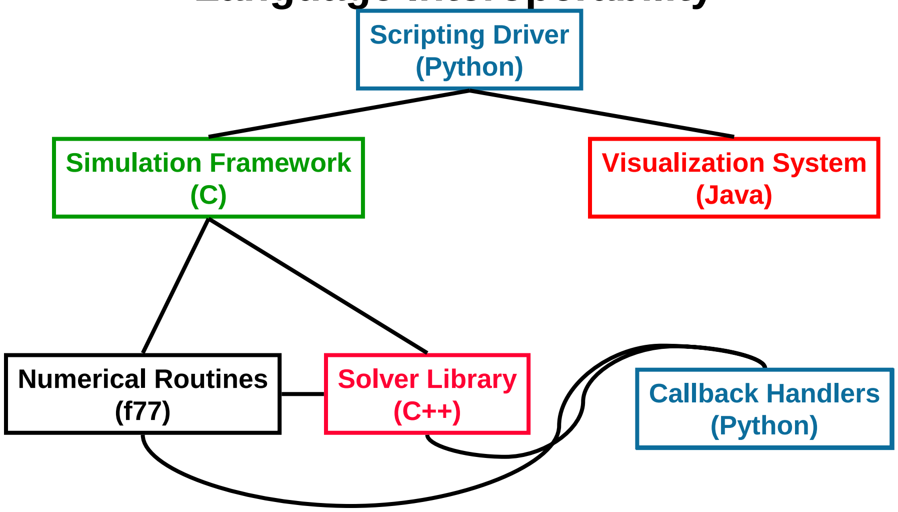
Goal of This Module

Legacy codes → Babelized CCA Components

- Introduction To:
  - Babel
  - SIDL
- See Babel in use
  - "Hello World" example
- Babel aspects of writing a CCA component

CCA
Common Component Architecture

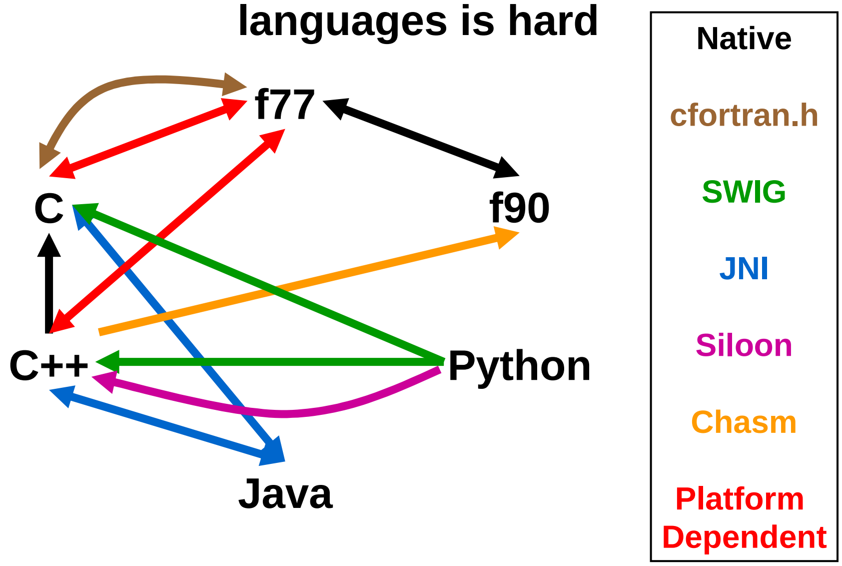
What I mean by "Language Interoperability"



69

CCA
Common Component Architecture

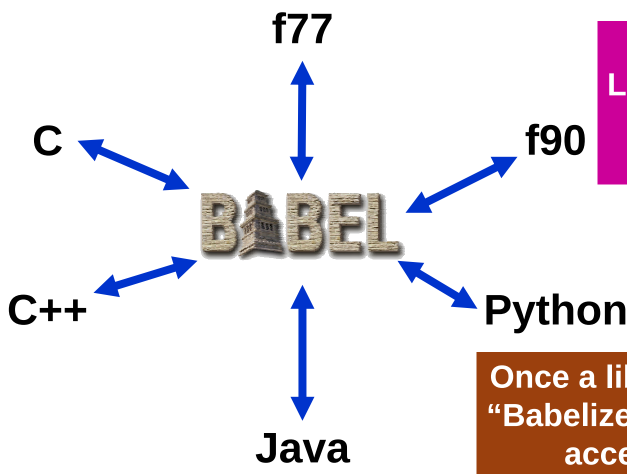
One reason why mixing languages is hard



70

CCA
Common Component Architecture

Babel makes all supported languages peers



This is not a Lowest Common Denominator Solution!

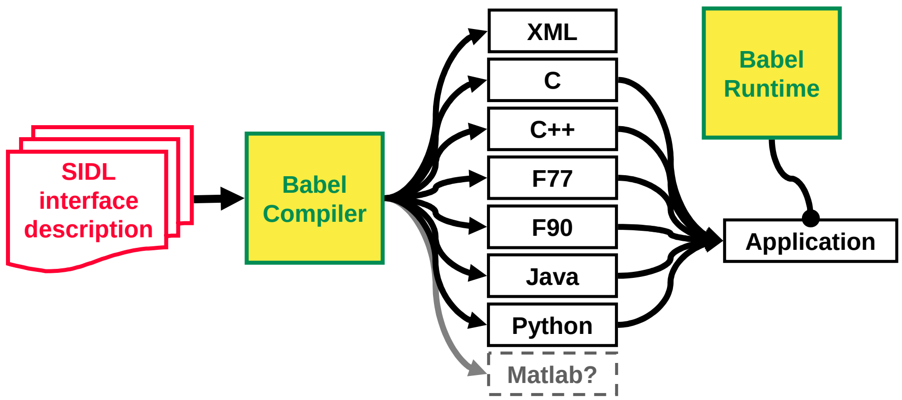
Once a library has been “Babelized” it is equally accessible from all supported languages

CCA
Common Component Architecture

Babel Module’s Outline

- Introduction
- Babel Basics
  - How to use Babel in a “Hello World” Example
  - SIDL Grammar
  - Wrapping legacy code
- Babel aspects of writing a CCA component

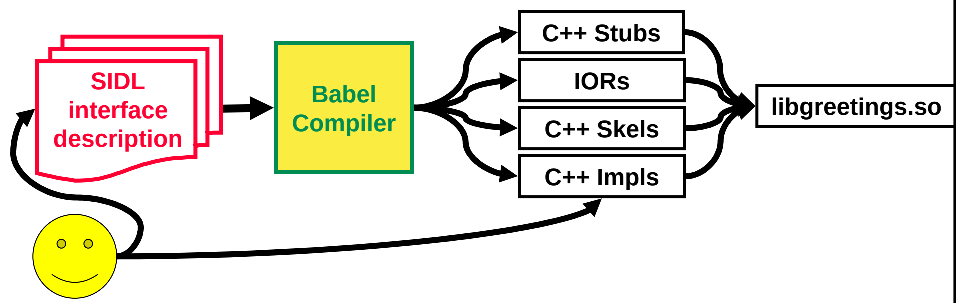
Babel's Two Parts: Code Generator + Runtime Library



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 {  }
}
```

Library Developer Does This...



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

75

Adding the Implementation

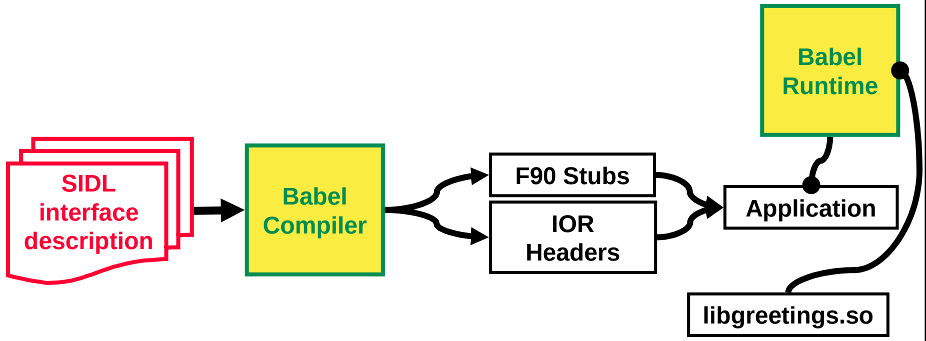
```
namespace greetings {
class English_impl {
  private:
    // DO-NOT-DELETE splicer.begin(greetings.English._impl)
    string d_name;
    // DO-NOT-DELETE splicer.end(greetings.English._impl)
```

```
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)
}
```

76

CCA Common Component Architecture

Library User Does This...



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

CCA Common Component Architecture

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

You'll use SIDL in the hands-on

- 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

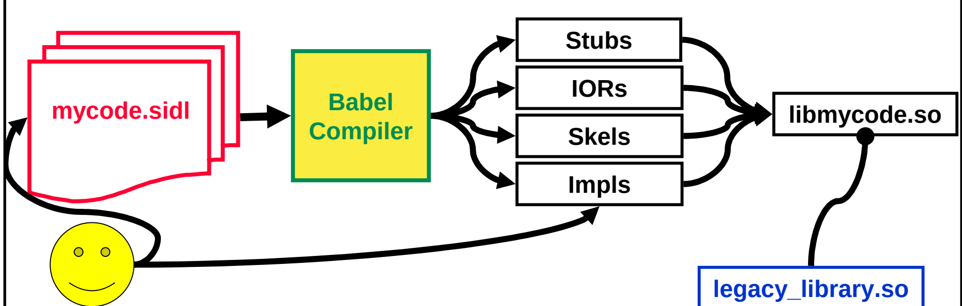
CCA Common Component Architecture

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

CCA Common Component Architecture

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

Supplementary material for notes

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

Tuning and Analysis Utilities

83

CCA Common Component Architecture

Supplementary material for notes

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)

84

CCA
Common Component Architecture

Babel Module's Outline

- Introduction
- Babel Basics
  - How to use Babel in a "Hello World" Example
  - SIDL Grammar
- Babel aspects of writing a CCA component

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)

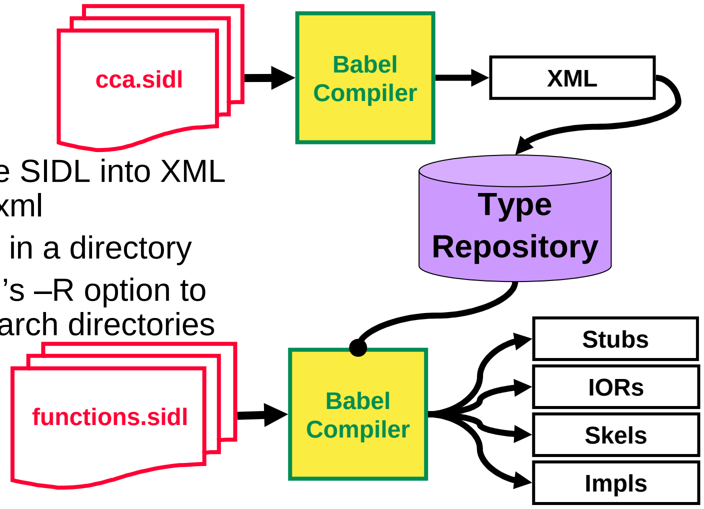
```
class LinearFunction implements functions.Function,
                                gov.cca.Component {
      double evaluate( in double x );
      void setServices( in cca.Services svcs );
}
```

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

CCA
Common Component Architecture

Supplementary material for notes

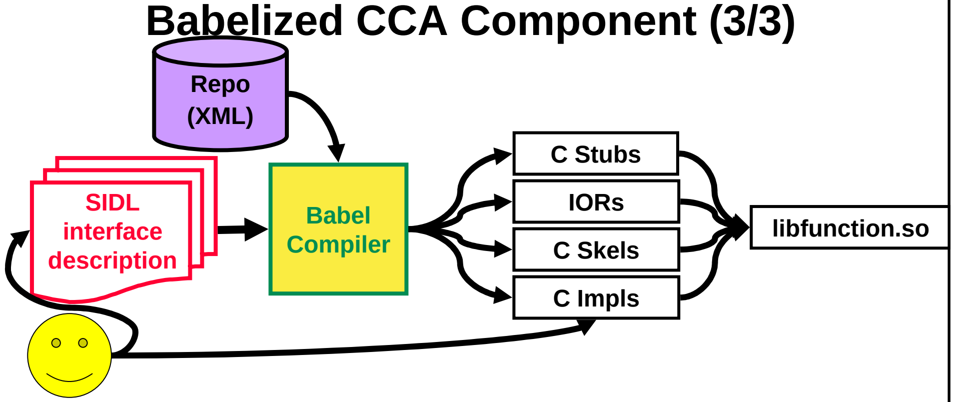
Tip: Use Babel's XML output like precompiled headers in C++



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

Supplementary material for notes

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

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

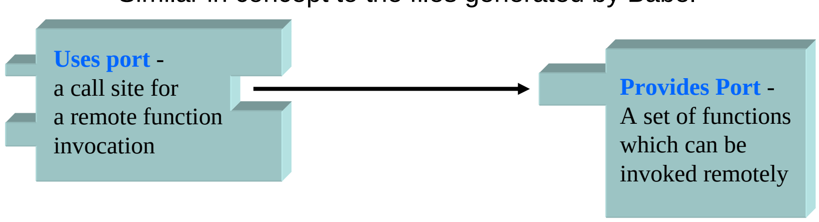
Distributed Computing with the CCA

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

CCA
Common Component Architecture

Component Composition

- Components can be linked along shared interfaces (ports) where one component invokes the services of another
  – Two types of Ports
    - Provides Ports – implements a remote interface
    - Uses Ports – uses a remote interface
  – A user and a provider of the same type can be linked
  – Details of run-time substrate shielded in stubs and skeletons
    - Similar in concept to the files generated by Babel



CCA Common Component Architecture

How Distributed Frameworks are Different

Remote Creation

- Launch components in remote address spaces
- Heterogeneity management
- Use resource managers to service requests on each remote resource
- Store, move and replicate component binaries

Remote Invocation

- Need global pointers and not local pointers
- Invoke methods across machine boundaries
- Need global namespace for names of components and services
- Mechanism for implementing remote method invocation (RMI)
- Introspection mechanisms to allow ports and services to be discovered and accessed

CCA Common Component Architecture

CCA Concepts that Influence Design of Distributed Frameworks (1)

- Ports
  - References to *provides* ports can move across address spaces
  - *Uses* ports are local to each component
- Services Object is present in each component
  - Manages all the ports
  - Hides details of framework-specific bindings for ports
- ComponentID: opaque handle to the component
  - Should be serializable and deserializable
  - Usually points to the services object

CCA
Common Component Architecture

CCA Concepts that Influence Design of Distributed Frameworks (2)

- Builder Service: charged with following operations
  - Create Components in remote address spaces
    - Return a ComponentID of instantiated components
    - Hide details of heterogeneous remote environments
  - Connect ports of components
    - Facilitate connection between uses and provides ports
      - Only if they are of the same SIDL type
    - Place provides port reference in the uses port table
- Introspection
  - Allow remote querying of a component
    - How many and what type of ports does the component have?

97

CCA
Common Component Architecture

Key Design Choices for Distributed CCA Frameworks (1)

- How is the CCA ComponentID represented in a distributed environment?
  - Handle that can be passed to remote components
  - Serialize and deserialize ComponentID
  - Belong to a namespace understood in the entire framework
  - Should enable optimized communication for co-located components
- How is the PortType represented?
  - *Provides* port should be designed as a remote service
  - *Uses* port should be a local object

98

CCA
Common Component Architecture

Key Design Choices for Distributed CCA Frameworks (2)

- Where can the key CCA functions be called from? Are they remote or local?
  - getPort() call on the services object
    - Should return a remote reference for provides ports
    - Note that the same call in the Ccaffeine framework returns a local object
  - Details of remote and local calls should be hidden
    - Framework should internally distinguish local and remote calls
- How can components be connected?
  - Need internal mechanism for uses port to obtain remote reference of the provides port
    - Information can be stored in a central table, facilitate development of GUIs to show component assembly
    - Distributed across components so they are aware of who they are connected to

99

CCA
Common Component Architecture

Supplementary material for handouts

Key Design Choices for Distributed CCA Frameworks (3)

- Should Builder Service be centralized or distributed?
  - A component can have its own builder service if
    - The builder service is lightweight
    - The components has special create/connect requirements

100

CCA
Common Component Architecture

Current CCA Distributed Frameworks

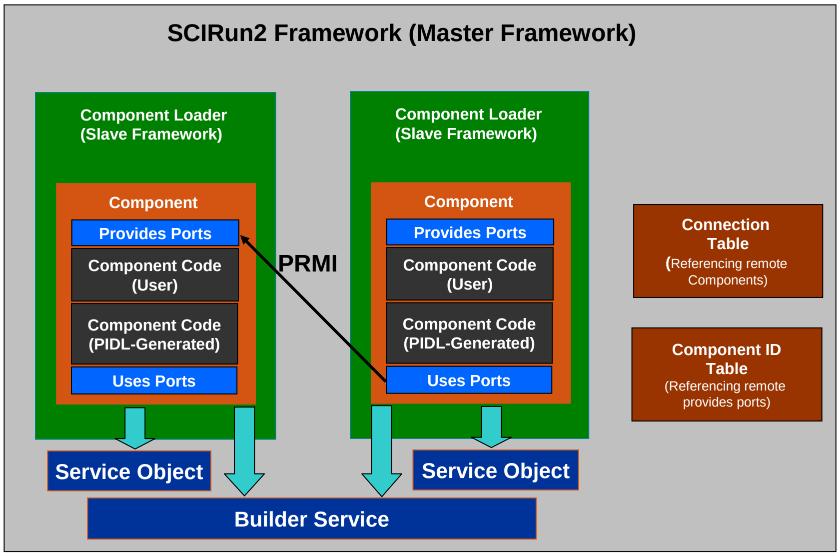
- SCIRun2
  - University of Utah
- LegionCCA
  - Binghamton University - State University of New York (SUNY)
- XCAT (Java and C++)
  - Indiana University and Binghamton University
- DCA
  - Indiana University
  - A research framework for MXN
- Frameworks address the design questions in different ways
  - Each has a different set of capabilities
  - Specialized for different kinds of applications

CCA
Common Component Architecture

SCIRun2

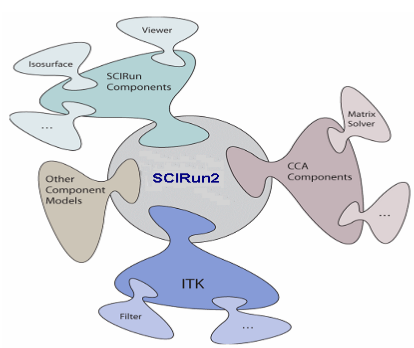
- Communication
  - C++ RMI that uses an in-house SIDL compiler
  - Co-location optimization
- Remote creation of distributed components
  - A slave framework resides in each remote address space
  - Uses *ssh* to start the slave framework
  - CCA BuilderService communicates with master framework which coordinates slave frameworks
- Support for distributed and parallel components
  - Can launch MPI–parallel components
    - Components interact via Parallel Remote Method Invocation
    - Each MPI process may contain multiple threads

Architecture of Distributed SCIRun2



SCIRun2 Meta-Component Model

- In the same way that components plug into a CCA framework, component models (such as CCA) plug into SCIRun2
- Allows components of several different families to be used together
- Currently supports: CCA (Babel), SCIRun Dataflow, Visualization Toolkit (Vtk); others coming…
- Bridging between components of different models is semi-automatic; current research is defining a more automatic form of bridging

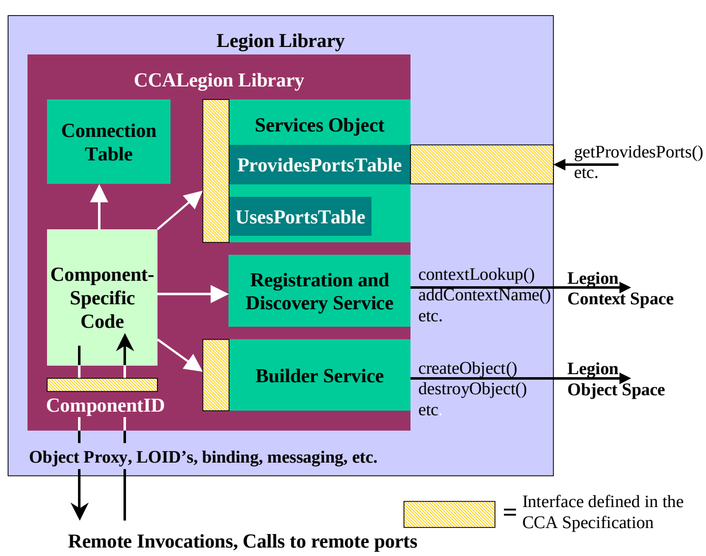


Research Area

LegionCCA

- Legion is a collection of software services for the Grid
  - Provides illusion of a virtual machine for geographically-distributed resources
- LegionCCA: models CCA components as Legion objects
- Component Communication
  - Uses Legion's built-in RPC mechanisms, based on Unix sockets
- ComponentID: based on Legion LOID
  - LOID: globally unique object id

Anatomy of a LegionCCA Component

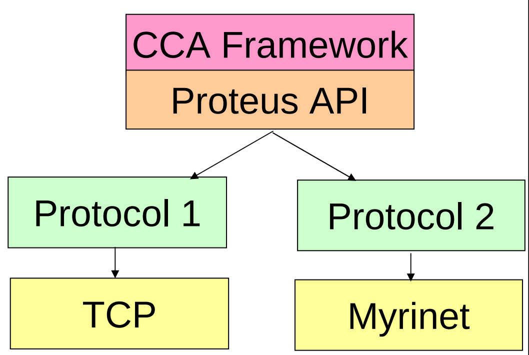


XCAT

- Based on Web Services Standards
  - Remote reference format is WSDL
  - Remote Communication is based on XSOAP
    - An implementation of the SOAP protocol from Indiana Univ.
- Remote creation of distributed components
  - Creation can currently be done via GRAM or SSH
    - GRAM: Grid Resource Allocation and Management
- XCAT-Java
  - Consistent with standards in Grid Web Services
- XCAT-C++
  - Uses Proteus for high performance remote communication
    - Proteus: multi-protocol library for messaging and RMI
    - Currently has two protocols: binary and SOAP

Proteus: Multi-Protocol Library

- One protocol does not suit all applications
- Proteus provides single-protocol abstraction to components
  - Allows users to dynamically switch between protocols
    - Example:Protocol1 & Protocol 2, in the picture
  - Facilitates use of specialized implementations of serialization and deserialization



CCA
Common Component Architecture

Babel RMI

Research!

- Allows Babel objects to be accessed through remote Babel stubs.
- Underlying RMI uses Proteus.
- Objects that can be transmitted (serializable) inherent from Serializable.
- Actual implementation of serialization functions is by users, since only they know what needs to be serialized.

CCA
Common Component Architecture

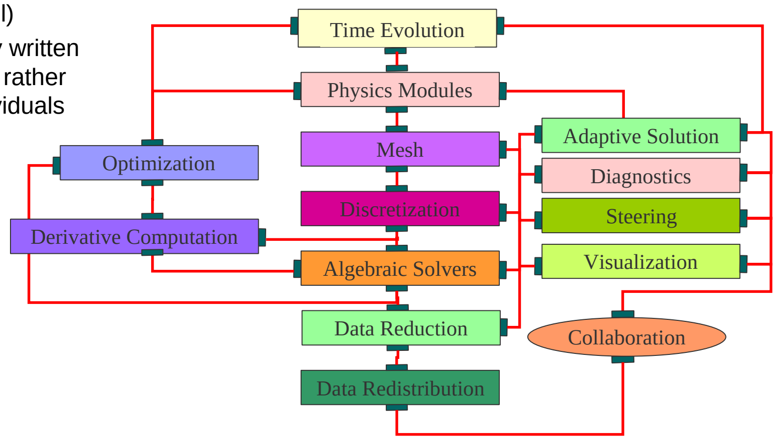
CCA Applications

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

CCA
Common Component Architecture

Modern Scientific Software Development

- Complex codes, often coupling multiple types of physics, time or length scales, involving a broad range of computational and numerical techniques
- Different parts of the code require significantly different expertise to write (well)
- Generally written by teams rather than individuals



CCA
Common Component Architecture

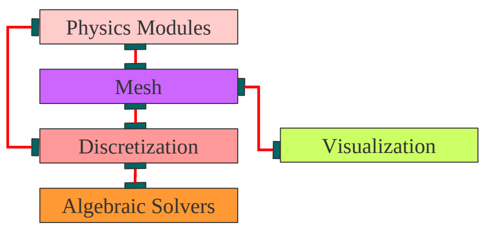
Overview

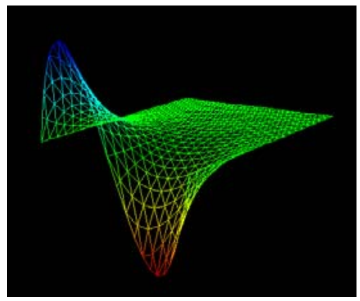
- Examples (scientific) of increasing complexity
  - Laplace equation
  - Time-dependent heat equation
  - Nonlinear reaction-diffusion system
  - Quantum chemistry
  - Climate simulation
- Tools
  - MxN parallel data redistribution
  - Performance measurement, modeling and scalability studies
- Community efforts & interface development
  - TSTT Mesh Interface effort
  - CCTTSS's Data Object Interface effort

CCA
Common Component Architecture

Laplace Equation

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

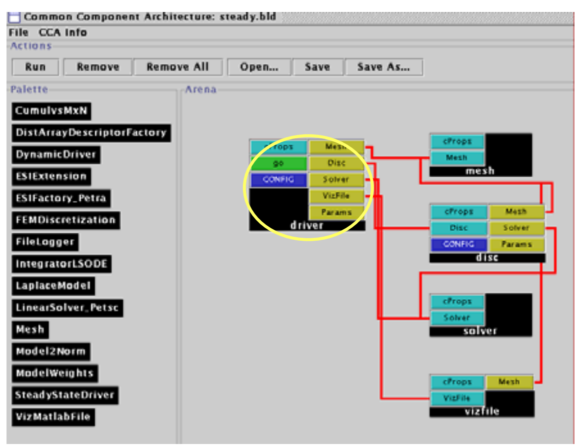




CCA
Common Component Architecture

Laplace Equation with Components

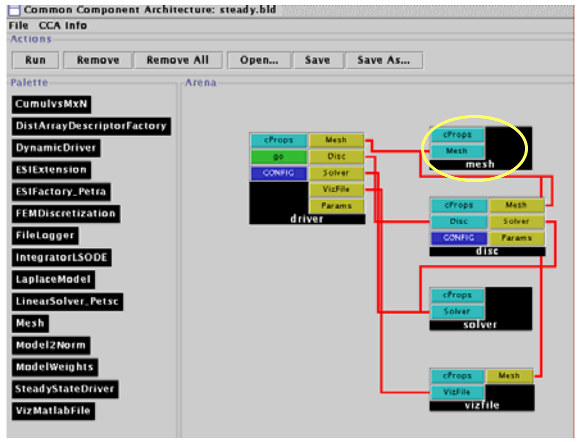
- The Driver Component
  - Responsible for the overall application flow
  - Initializes the mesh, discretization, solver and visualization components
  - Sets the physics parameters and boundary condition information



CCA
Common Component Architecture

Laplace Equation with Components

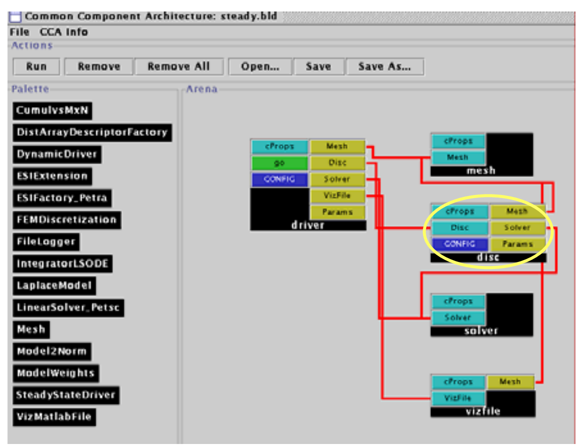
- The Driver
- The Mesh Component
  - Provides geometry, topology, and boundary information
  - Provides the ability to attach user defined data as tags to mesh entities
  - Is used by the driver, discretization and visualization components



CCA
Common Component Architecture

Laplace Equation with Components

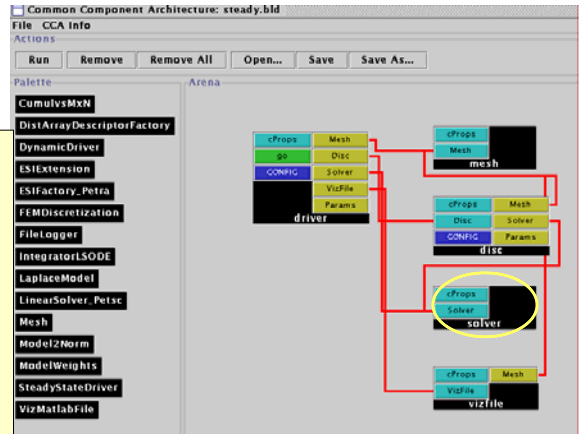
- The Driver
- The Mesh
- The Discretization Component
  - Provides a finite element discretization of basic operators (gradient, Laplacian, scalar terms)
  - Driver determines which terms are included and their coefficients
  - Boundary conditions, assembly etc



CCA Common Component Architecture

Laplace Equation with Components

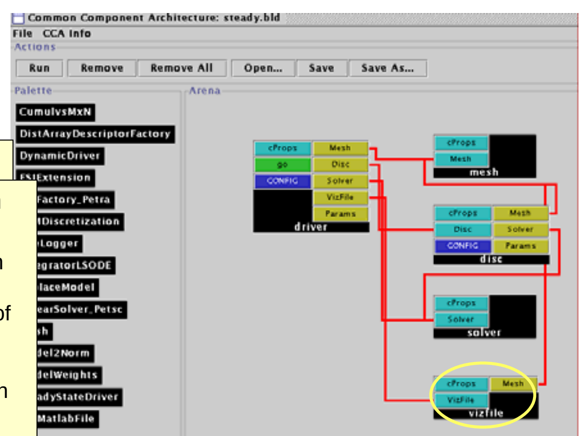
- The Driver
- The Mesh
- The Discretization
- The Solver Component
  - Provides access to vector and matrix operations (e.g., create, destroy, get, set)
  - Provides a “solve” functionality for a linear operator



CCA Common Component Architecture

Laplace Equation with Components

- The Driver
- The Mesh
- The Discretization
- The Solver
- 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



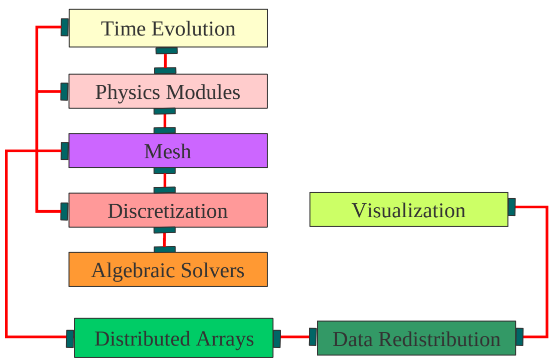
Time-Dependent Heat Equation

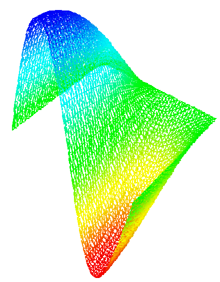
$$\delta\varphi/\delta t = \nabla^2\varphi \quad (x,y,t) \in [0,1] \times [0,1]$$

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

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

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





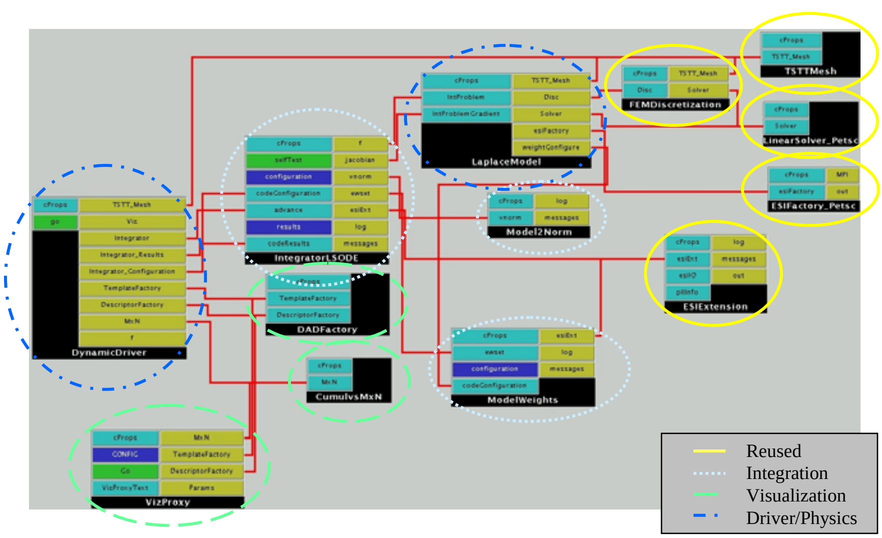
Some things change…

- Requires a time integration component
  - Based on the LSODE library
- Uses a new visualization component
  - Based on AVS
- The visualization component requires a Distributed Array Descriptor component
  - Similar to HPF arrays
- The driver component changes to accommodate the new physics

… 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

Heat Equation Wiring Diagram



What did this exercise teach us?

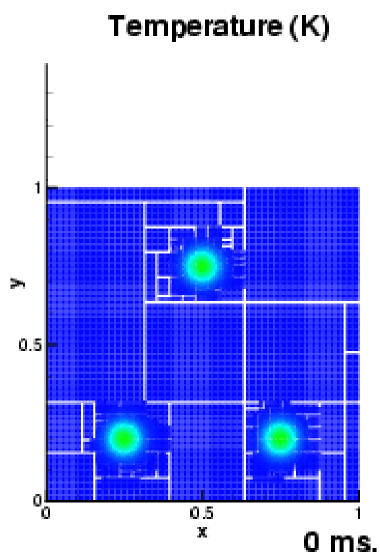
- Easy to incorporate the functionalities of components developed at other labs and institutions given a well-defined interface.
  - In fact, some components (one uses and one provides) were developed simultaneously across the country from each other after the definition of a header file.
  - Amazingly 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

Nonlinear Reaction-Diffusion 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)



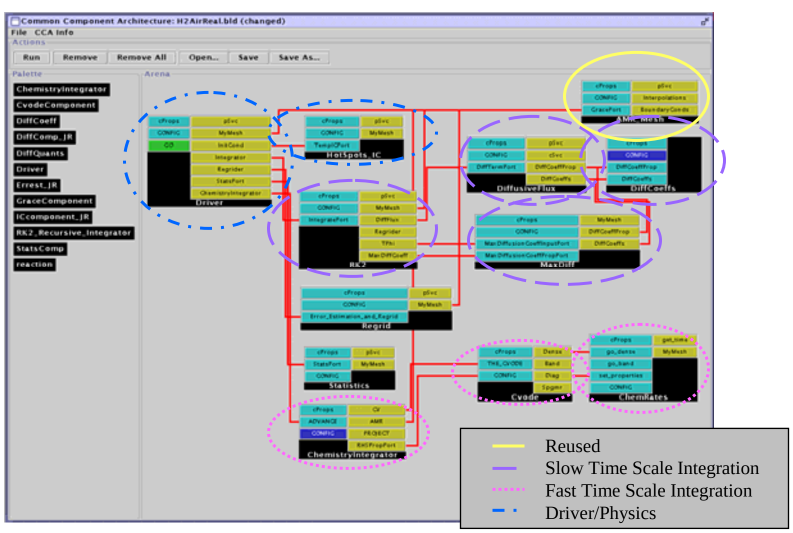
CCA Common Component Architecture

Numerical Solution

- Adaptive Mesh Refinement: GrACE
- Stiff integrator: CVODE
- Diffusive integrator: 2^{nd} Order Runge Kutta
- Chemical Rates: legacy f77 code
- Diffusion Coefficients: legacy f77 code
- New code less than 10%

CCA Common Component Architecture

Reaction-Diffusion Wiring Diagram

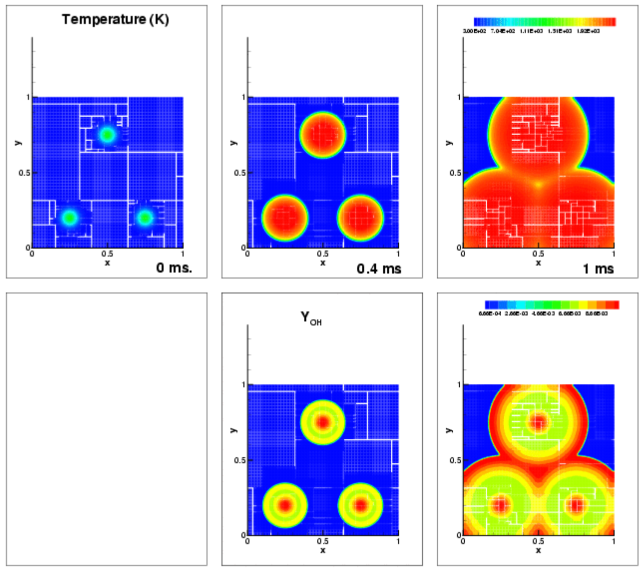


CCA
Common Component Architecture

Evolution of the Solution

Temperature

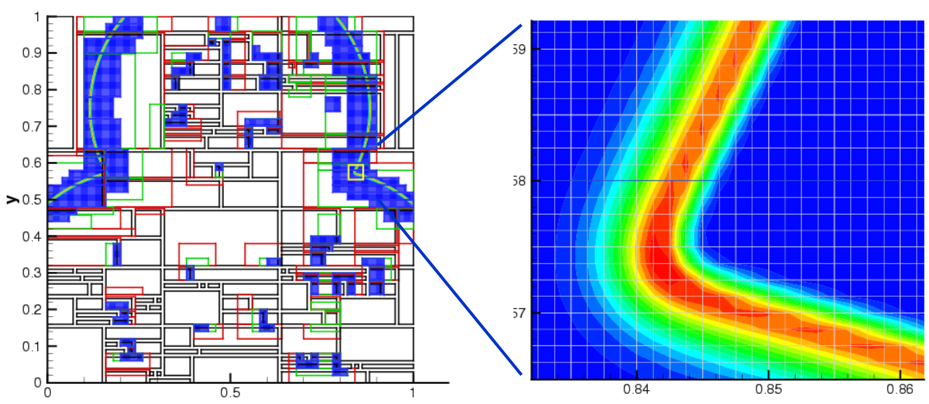
OH Profile



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

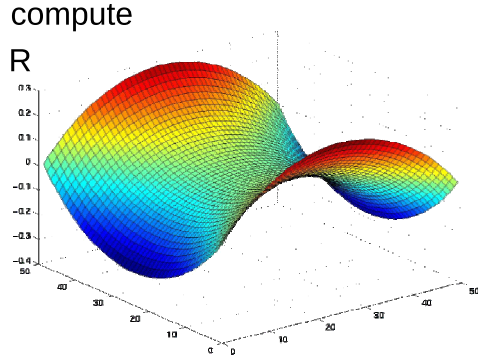


CCA
Common Component Architecture

Unconstrained Minimization Problem

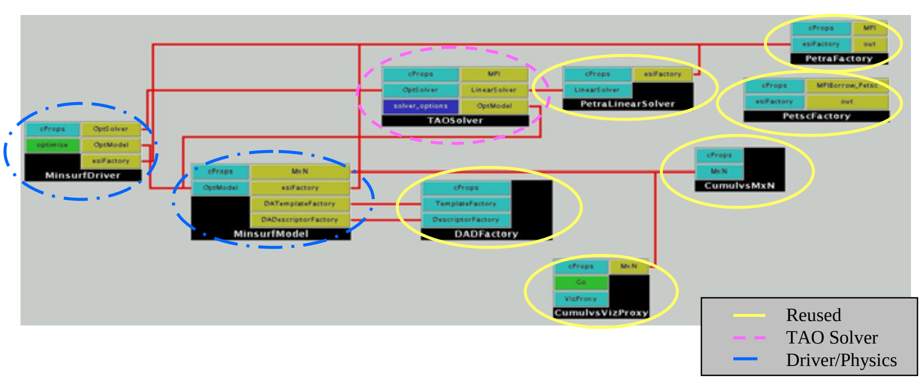
- Given a rectangular 2-dimensional domain and boundary values along the edges of the domain
- Find the surface with minimal area that satisfies the boundary conditions, i.e., compute

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



CCA
Common Component Architecture

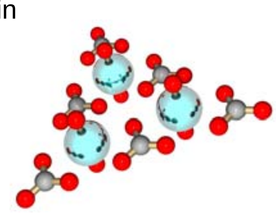
Unconstrained Minimization Using a Structured Mesh



CCA
Common Component Architecture

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

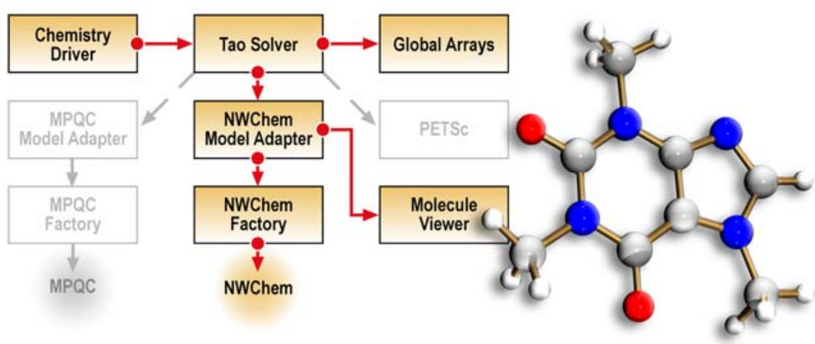


131

CCA
Common Component Architecture

Molecular Optimization Overview

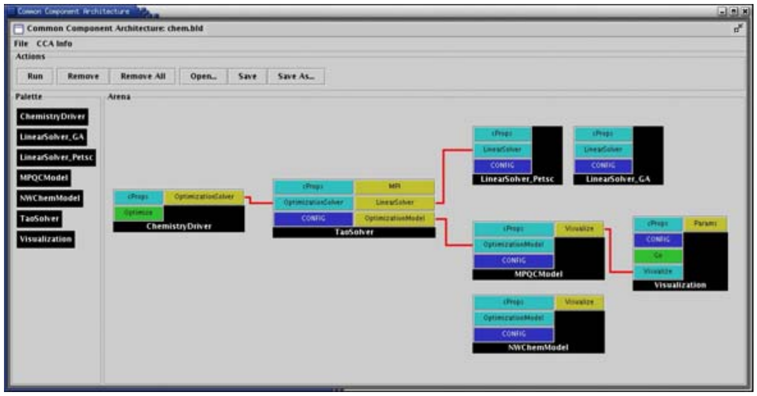
- Decouple geometry optimization from electronic structure
- Demonstrate interoperability of electronic structure components
- Build towards more challenging optimization problems, e.g., protein/ligand binding studies



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

132

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

Actual Improvements

Molecule	NWChem	NWChem/TAO	MPQC	MPQC/TAO
Glycine	33	19	26	19
Isoprene	56	45	75	43
Phosposerine	79	67	85	62
Aspirin	43	51	54	48
Cholesterol	33	30	27	30

Function and gradient evaluations

CCA
Common Component Architecture

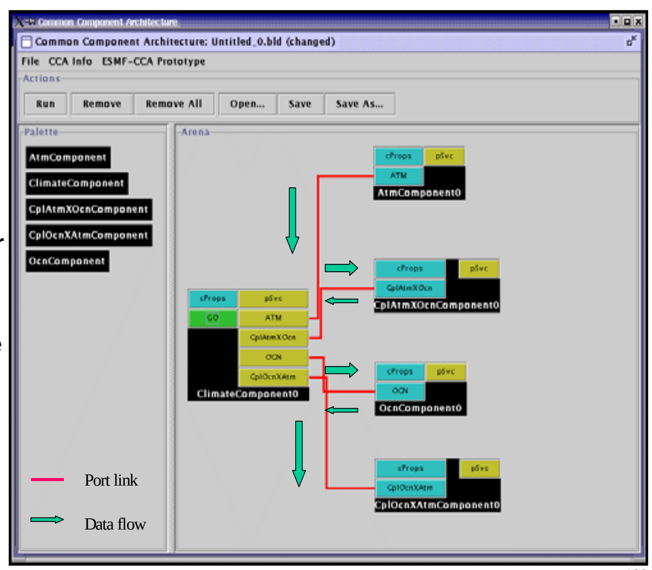
Componentized Climate Simulations

- 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

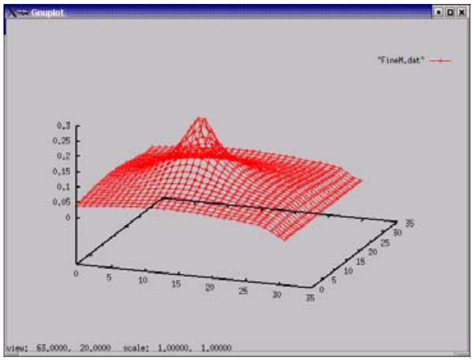
Coupled Atmosphere-Ocean Model Assembly

- **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



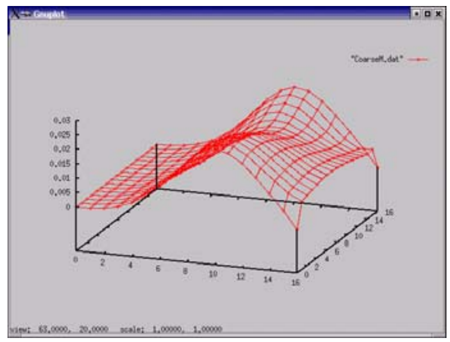
CCA
Common Component Architecture

Simulation Results



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

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



137

CCA
Common Component Architecture

Concurrency At Multiple Granularities

- Certain simulations need multi-granular concurrency
  - Multiple Component Multiple Data, multi-model runs
- Usage Scenarios:
  - Model coupling (e.g. Atmosphere/Ocean)
  - General multi-physics applications
  - Software licensing issues

Driver		
Atmosphere	Ocean	Land
Coupler		

- 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
    - Link as components (through AbstractFramework service; highly experimental at present)

138

CCA
Common Component Architecture

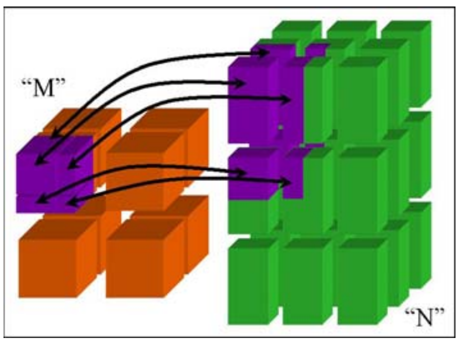
Overview

- Examples (scientific) of increasing complexity
  – Laplace equation
  – Time-dependent heat equation
  – Nonlinear reaction-diffusion system
  – Quantum chemistry
  – Climate simulation
- Tools
  – MxN parallel data redistribution
  – Performance measurement, modeling and scalability studies
- Community efforts & interface development
  – TSTT Mesh Interface effort
  – CCTTSS's Data Object Interface effort

CCA
Common Component Architecture

"MxN" Parallel Data Redistribution: The Problem…

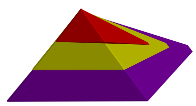
- Create complex scientific simulations by coupling together multiple parallel component models
  – Share data on "M" processors with data on "N"
    - M != N ~ Distinct Resources (Pronounced "M by N")
  – Model coupling, e.g., climate, solver / optimizer
  – Collecting data for visualization
    - Mx1; increasingly MxN (parallel rendering clusters)
- Define common 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/

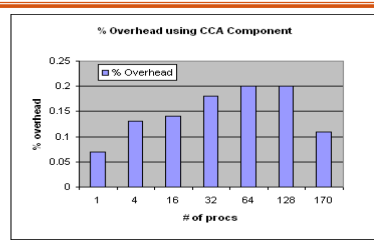
CCA
Common Component Architecture

CCA Delivers Performance

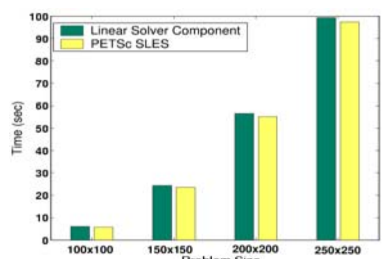
Local
- No CCA overhead within components
- Small overhead between components
- Small overhead for language interoperability
- Be aware of costs & design with them in mind
  - Small costs, easily amortized

Parallel
- No CCA overhead on parallel computing
- Use your favorite parallel programming model
- Supports SPMD and MPMD approaches

Distributed (remote)
- No CCA overhead – performance depends on networks, protocols
- CCA frameworks support OGSA/Grid Services/Web Services and other approaches



Maximum 0.2% overhead for CCA vs native C++ code for parallel molecular dynamics up to 170 CPUs



Aggregate time for linear solver component in unconstrained minimization problem w/ PETSc

CCA Common Component Architecture

Overhead from Component Invocation

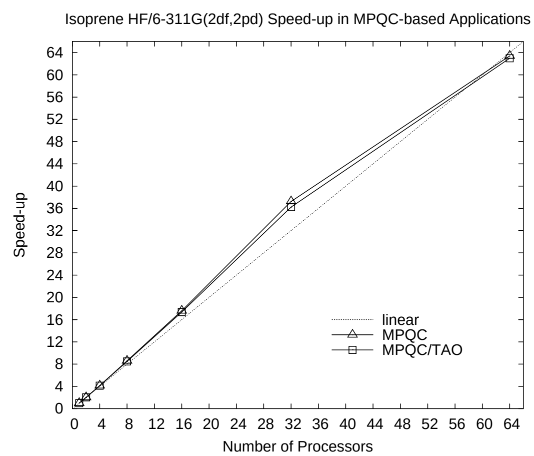
- Invoke a component with different arguments
  - Array
  - Complex
  - Double Complex
- Compare with f77 method invocation
- Environment
  - 500 MHz Pentium III
  - Linux 2.4.18
  - GCC 2.95.4-15
- Components took 3X longer
- Ensure granularity is appropriate!
- Paper by Bernholdt, Elwasif, Kohl and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

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

CCA Common Component Architecture

Scalability : Component versus Non-component. I

- Quantum chemistry simulation
- Sandia's MPQC code
  - Both componentized and non-componentized versions
- Componentized version used TAO's optimization algorithms
- Problem :Structure of isoprene HF/6-311G(2df,2pd)

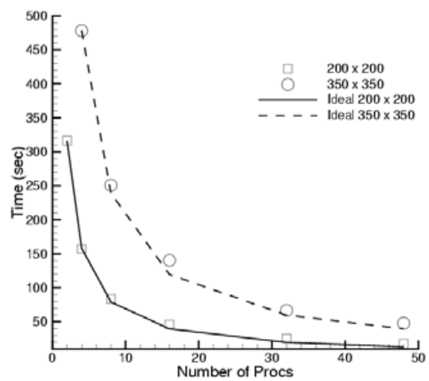


Parallel Scaling of MPQC w/ native and TAO optimizers

CCA Common Component Architecture

Scalability : Component versus Non-component. II

- Hydrodynamics; uses CFRFS set of components
- Uses GrACEComponent
- 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.

CCA Common Component Architecture

Performance Measurement In A Component World

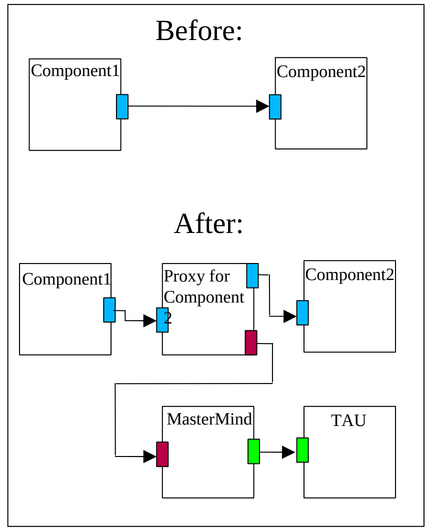
- CCA provides a novel means of profiling & modeling component performance
- Need to collect incoming inputs and match them up with the corresponding performance, but how ?
  - Need to “instrument” the code
    - But has to be non-intrusive, since we may not “own” component code
- What kind of performance infrastructure can achieve this?
  - Previous research suggests proxies
    - Proxies serve to intercept and forward method calls

CCA
Common Component Architecture

"Integrated" Performance Measurement Capability

Measurement infrastructure:

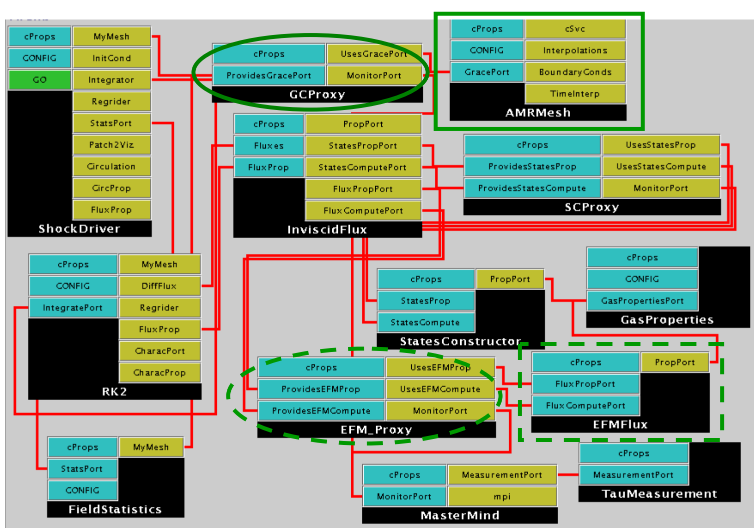
- **Proxy**
  - Notifies MasterMind of all method invocations of a given component, along with performance dependent inputs
  - Generated automatically using PDT
- **MasterMind**
  - Collects and stores all measurement data
- **TAU**
  - Makes all performance measurements



Tuning and Analysis Utilities

CCA
Common Component Architecture

Component Application With Proxies



CCA Common Component Architecture

Overview

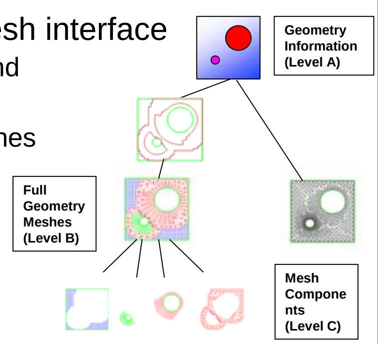
- Examples (scientific) of increasing complexity
  - Laplace equation
  - Time-dependent heat equation
  - Nonlinear reaction-diffusion system
  - Quantum chemistry
  - Climate simulation
- Tools
  - MxN parallel data redistribution
  - Performance measurement, modeling and scalability studies
- Community efforts & interface development
  - TSTT Mesh Interface effort
  - CCTTSS's Data Object Interface effort

CCA Common Component Architecture

The Next Level

TSTT

- 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

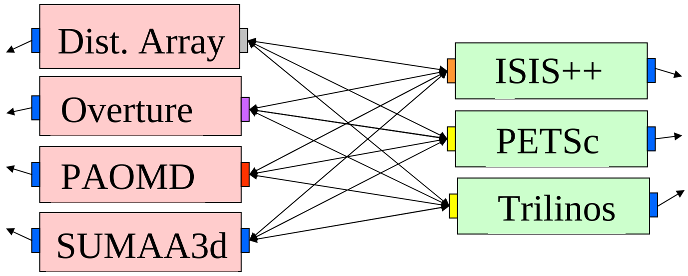


CCA Common Component Architecture

Proliferations of interfaces – the N^2 problem

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

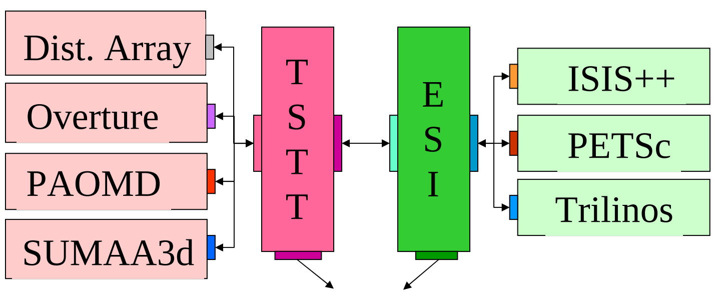


CCA Common Component Architecture

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

CCTTSS Research Thrust Areas and Main Working Groups

- Scientific Components
  Lois Curfman McInnes, ANL (curfman@mcs.anl.gov)
- "MxN" Parallel Data Redistribution
  Jim Kohl, ORNL (kohlja@ornl.gov)
- Frameworks
  - Language Interoperability / Babel / SIDL
  Gary Kumfert, LLNL (kumfert@llnl.gov)
- User Outreach
  David Bernholdt, ORNL (bernholdtde@ornl.gov)

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