



Welcome to the Common Component Architecture Tutorial

CCA Forum Tutorial Working Group

[http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org](http://www.cca-forum.org/tutorials/tutorial-wg@cca-forum.org)



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 N² 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 (CCTSS, 2001)
 - Part of New DOE Scientific Simulation Program
 - Technology Development in Several Thrust Areas...

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/>

3

Center for Component Technology for Terascale Simulation Software (CCTSS)

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

Lawrence Livermore
National LaboratoryLos Alamos
National Laboratoryoml
Office of Mathematical, Information and Computational SciencesPacific Northwest
National LaboratorySandia
National
LaboratoriesTHE
UNIVERSITY
OF UTAH

Lead PI: Rob Armstrong, SNL <rob@sandia.gov>

<http://www.cca-forum.org/cctss/>

4



CCTSS 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
Scott Kohn, LLNL (skohn@llnl.gov)
- **User Outreach**
David Bernholdt, ORNL (bernholdtde@ornl.gov)

5

Acknowledgements

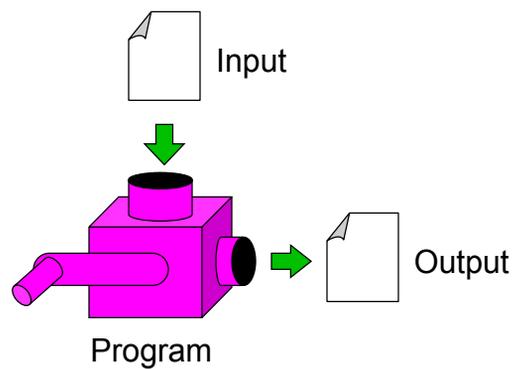
- **CCA Forum Tutorial WG**
 - Rob Armstrong, David Bernholdt, Wael Elwasif, Lori Freitag, Dan Katz, Jim Kohl, Gary Kurfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, Torsten Wilde
 - ANL, JPL, LANL, LLNL, ORNL, SNL
- **And many more contributing to CCA itself...**
 - ANL - Lori Freitag, 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 - Tom Epperly, Scott Kohn, Gary Kurfert, ...
 - ORNL - David Bernholdt, Wael Elwasif, Jim Kohl, Torsten Wilde, ...
 - PNNL - Jarek Nieplocha, Theresa Windus, ...
 - SNL - Rob Armstrong, Ben Allan, Lori Freitag, Curt Janssen, Jaideep Ray, ...
 - University of Utah - Steve Parker, ...
 - And others as well ...

6

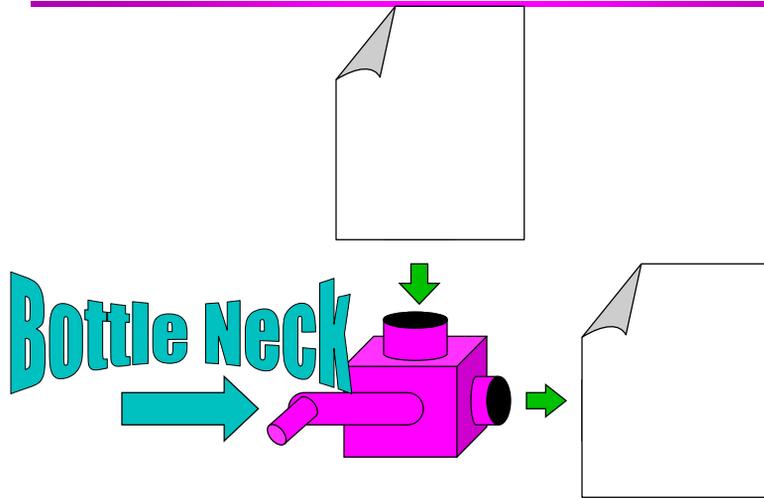
Next: The Sausage Grinder Talk

A Pictorial Introduction to Components in Scientific Computing

Once upon a time...

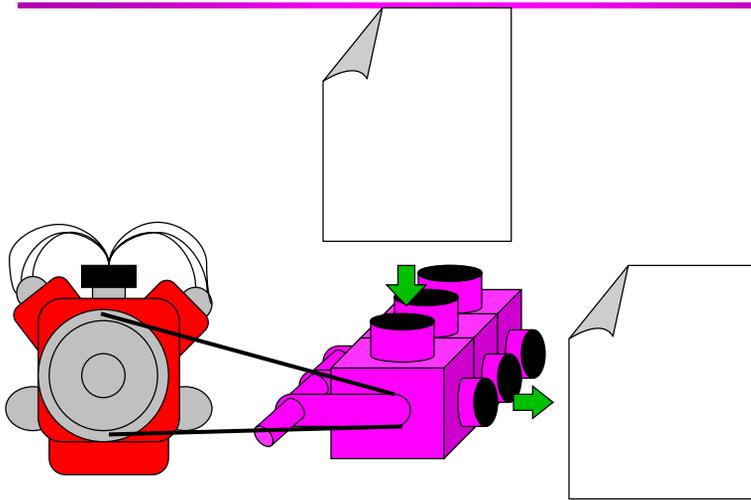


As Scientific Computing grew...



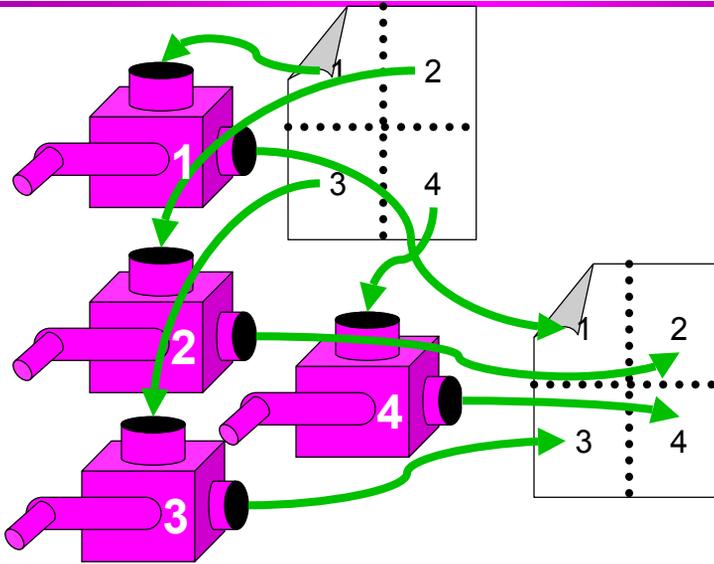
3

Tried to ease the bottle neck



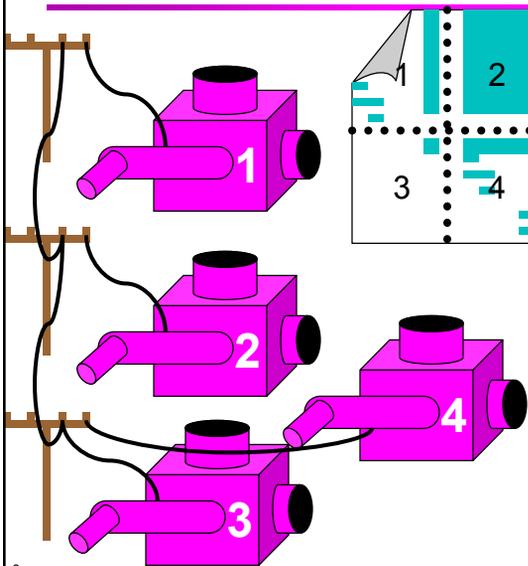
4

SPMD was born.



5

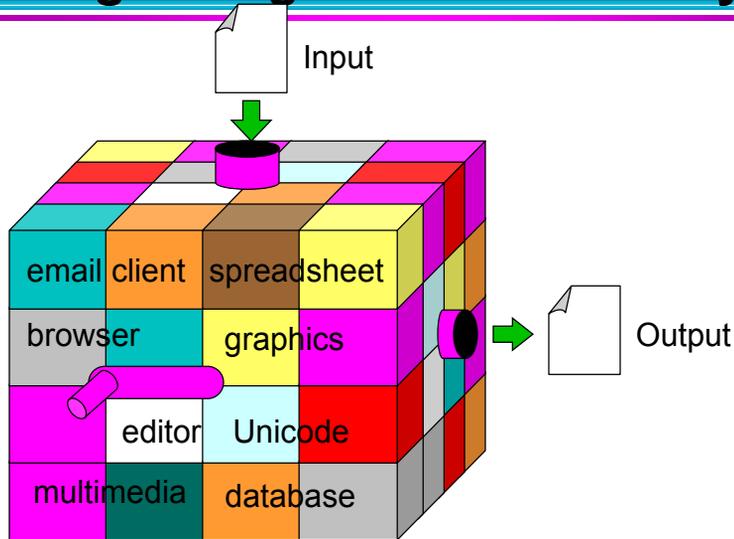
SPMD worked.



But it isn't easy!!!

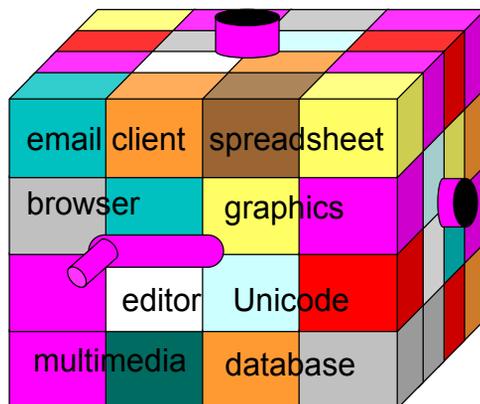
6

Meanwhile, corporate computing was growing in a different way



7

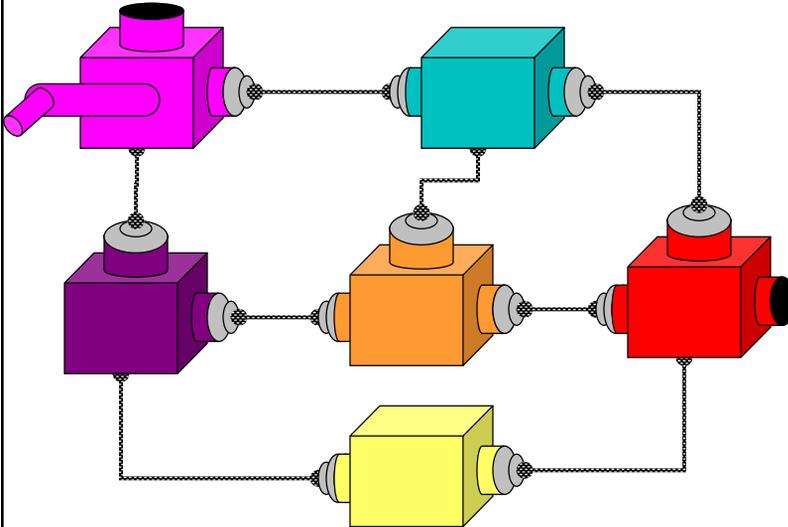
This created a whole new set of problems → complexity



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

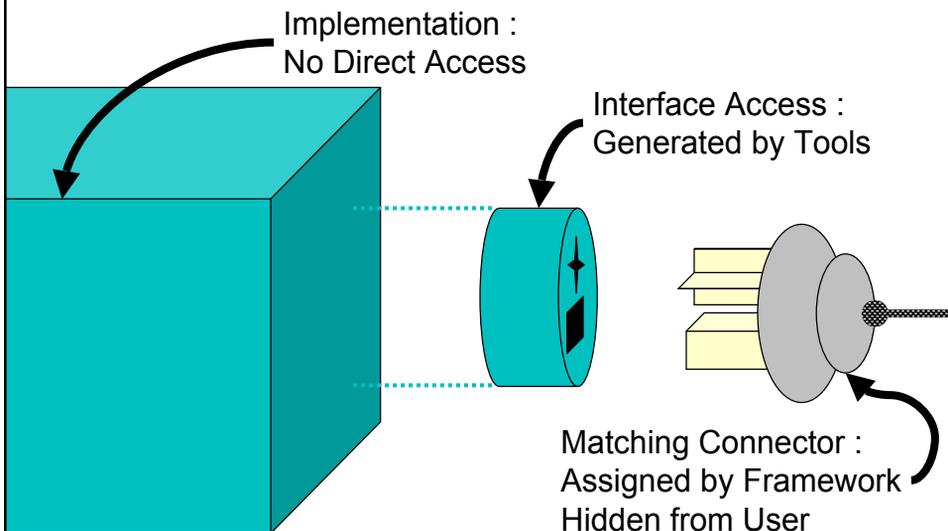
8

Component Technology addresses these problems



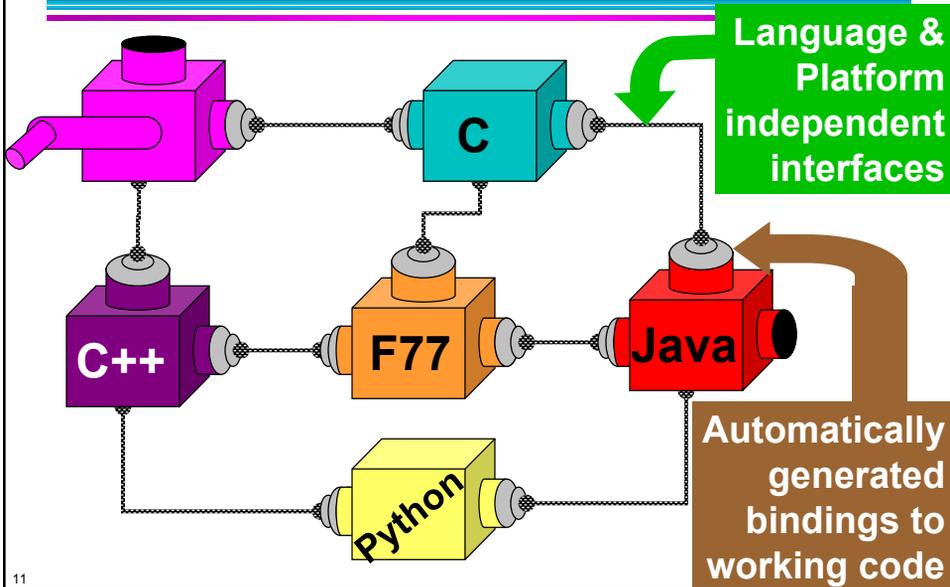
9

So what's a component ???

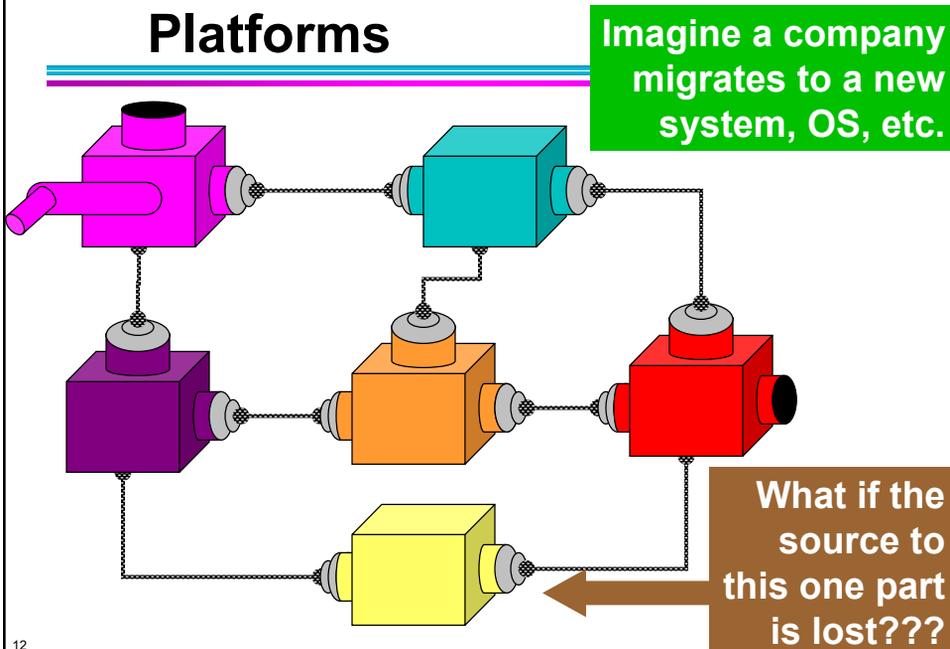


10

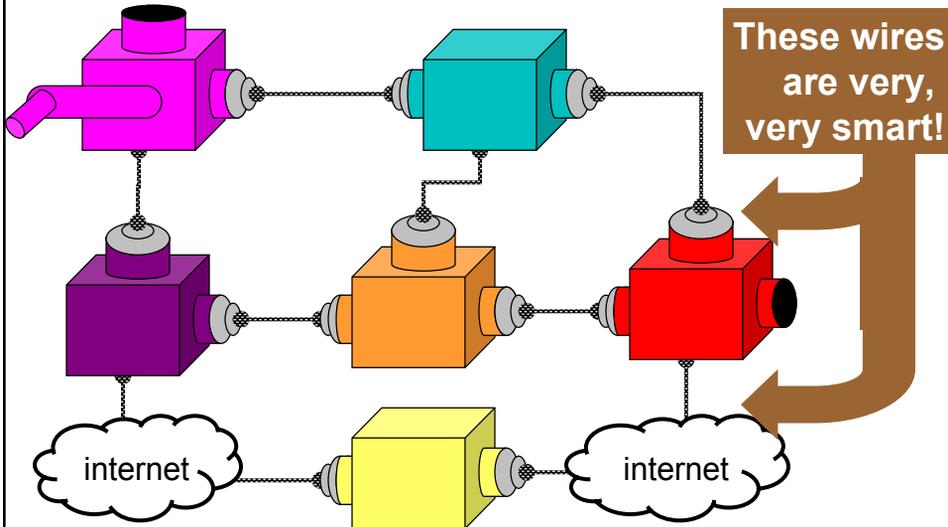
1. Interoperability across multiple languages



2. Interoperability Across Multiple Platforms

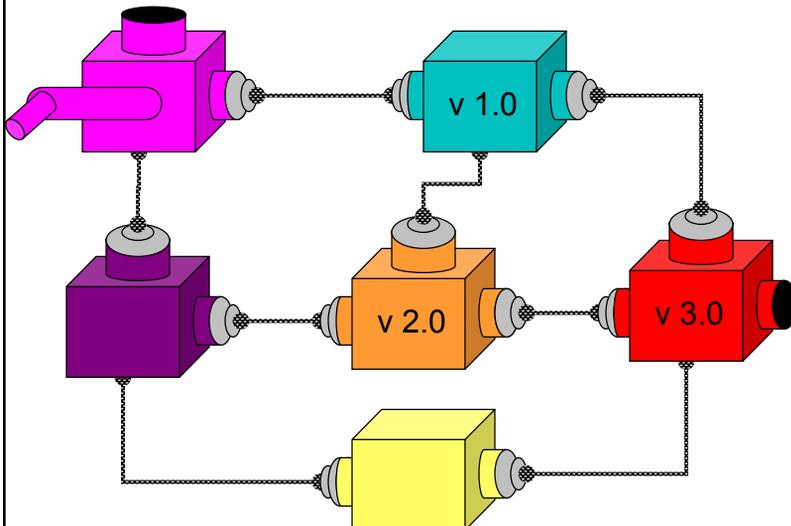


Transparent Distributed Computing



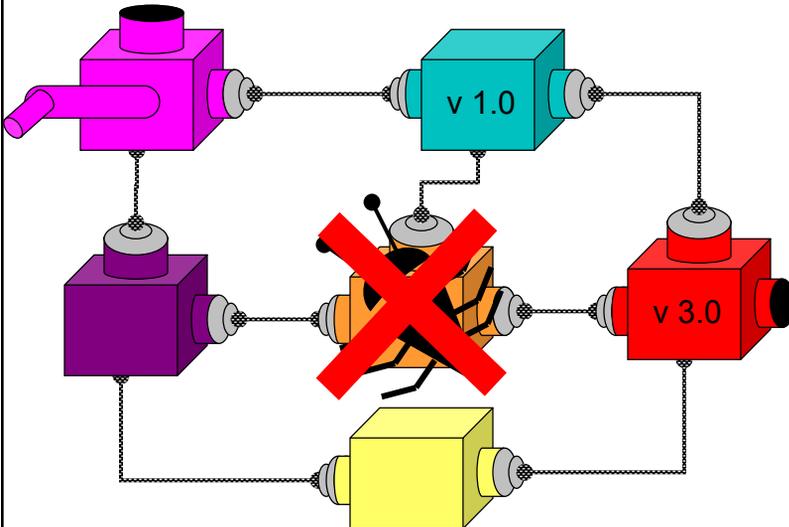
13

3. Incremental Evolution With Multiple 3rd party software



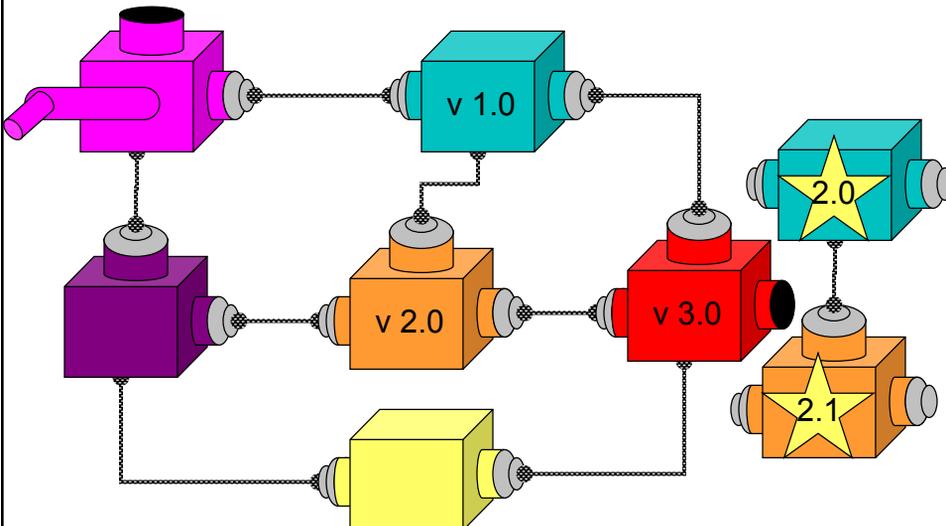
14

Now suppose you find this bug...



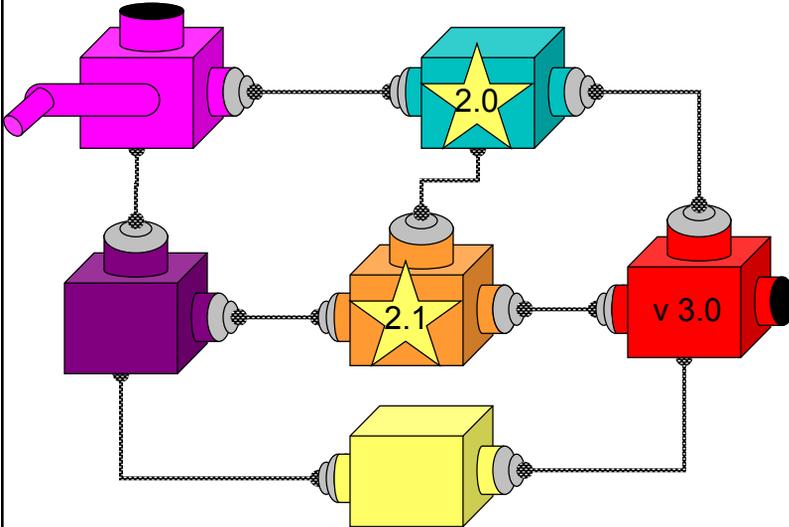
15

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



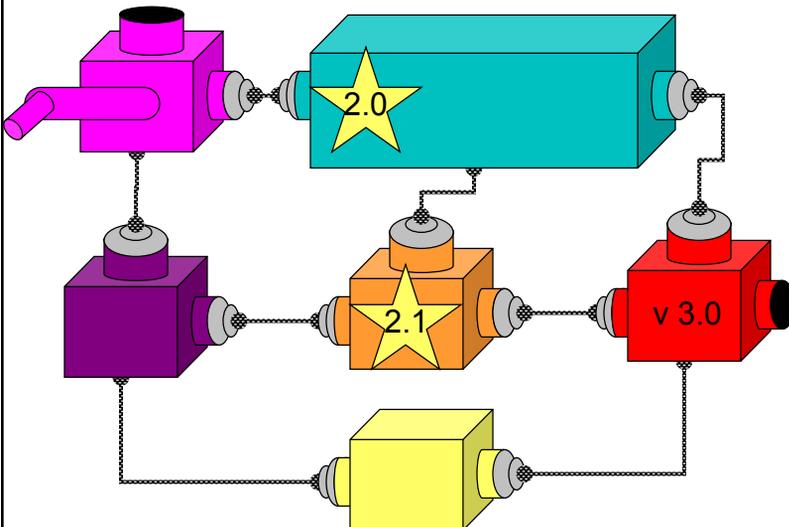
16

Great News: Solvable with Components



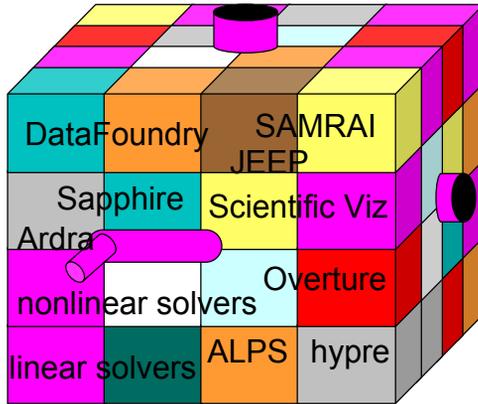
17

Great News: Solvable with Components



18

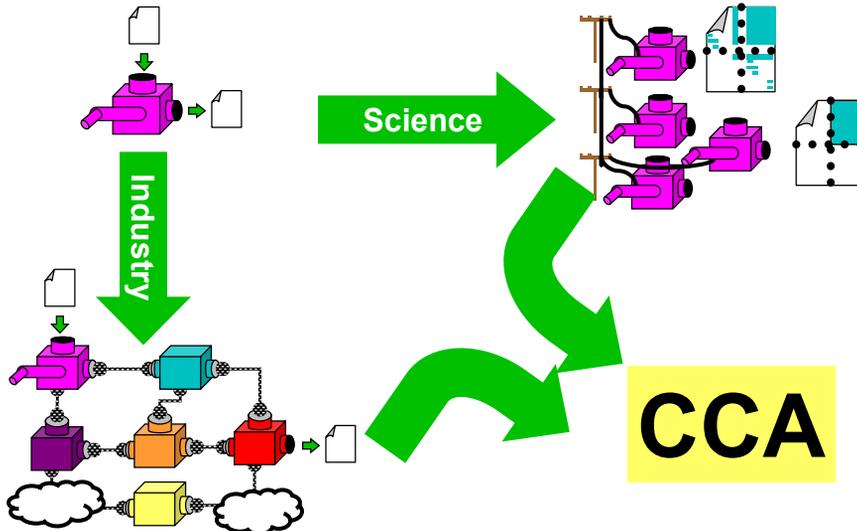
Why Components for Scientific Computing → Complexity



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

19

The Model for Scientific Component Programming



20



The End

Next: [Intro to Components](#)



Introduction to Components

CCA Forum Tutorial Working Group

<http://www.cca-forum.org/tutorials/>
tutorial-wg@cca-forum.org



Overview

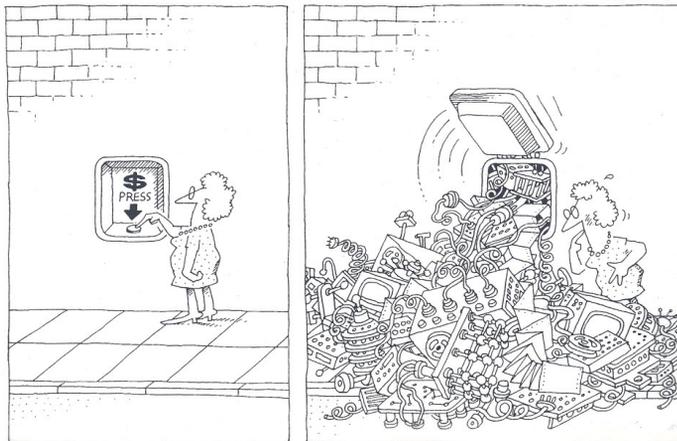
- **Why** do we need components?
- **What** are components?
- **How** do we make components?

Why Components

- In “Components, The Movie”
 - Interoperability across multiple languages
 - Interoperability across multiple platforms
 - Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)
- Complexity

3

Why Components



The task of the software development team is to engineer the illusion of simplicity [Booch].

4

Software Complexity

- Software crisis
 - “Our failure to master the complexity of software results in projects that are late, over budget, and deficient in their stated requirements.” [Booch]
- Can’t escape it
 - “The complexity of software is an essential property, not an accidental one.” [Brooks]
- Help is on the way...
 - “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]
 - “Intracomponent linkages are generally stronger than intercomponent linkages.” [Simon]
 - “Frequently, complexity takes the form of a hierarchy.” [Courtois]

5

The Good the Bad and the Ugly

- An example of what can lead to a crisis in software:
- 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
- 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!).

6

Components Promote Reuse



Hero programmer producing single-purpose, monolithic, tightly-coupled parallel codes

- Components promote software reuse
 - “The best software is code you don’t have to write”
[Steve Jobs]
- Reuse, through cost amortization increases software quality
 - thoroughly tested code
 - highly optimized code
 - improved support for multiple platforms
 - developer team specialization

7

What Are Components

- **Why** do we need components?
- **What** are components?
- **How** do we make components?

8

What Are Components [Szyperski]

- A component is a binary unit of independent deployment
 - well separated from other components
 - fences make good neighbors
 - can be deployed independently
- A component is a unit of third-party composition
 - is composable (even by physicists)
 - comes with clear specifications of what it requires and provides
 - interacts with its environment through well-defined interfaces
- A component has no persistent state
 - temporary state set only through well-defined interfaces
 - throw away that dependence on global data (common blocks)
- Similar to Java packages and Fortran 90 modules (with a little help)

9

What Does This Mean

- So what does this mean
 - Components are “plug and play”
 - Components are reusable
 - Component applications are evolvable

10

Component Forms *[Cheesman & Daniels]*

- Component Standard
 - must conform to some sort of environment standard (Framework)
- Component Specification
 - specification of what a component does
- Component Interface
 - specification of procedure names and procedure parameters
- Component Implementation
 - written in a computer language (Fortran for example)
- Installed Component
 - a shared object library (.so file)
- Component Object
 - services and state joined together

11

What is a Component Architecture

- A set of standards that allows:
 - Multiple groups to write units of software (components)
 - The groups to be sure that their components will work with other components written in the same architecture
- A framework that holds and runs the components
 - And provides services to the components to allow them to know about and interact with other components

12

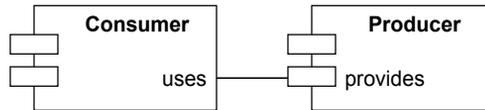
What Are Components II

- Components live in **an environment** and interact with the environment through a framework and connections with other components.
- Components can **discover information** about their environment from the framework.
- Components must explicitly publish what capabilities they **provide**.
- Components must explicitly publish what connections they **require**.
- Components are a runtime entity.

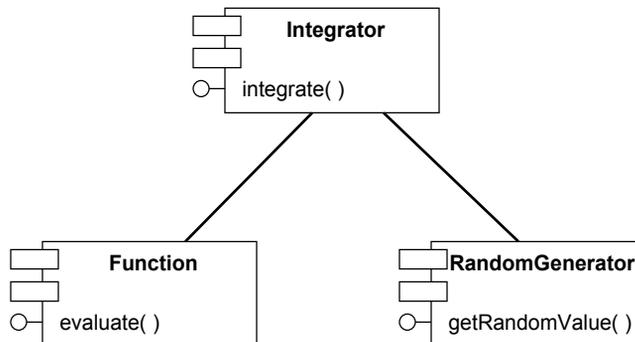
Components Are Different From Objects

- You can build components out of object classes.
 - (or out of Fortran procedures)
- But a component is more than just an object.
- A component only exists in the context of a Component Standard (Framework).

Pictorial Example



Three Components



How Do We Make Components

- **Why** do we need components?
- **What** are components?
- **How** do we make components?

17

Interface Declaration



Integrator.h

```
class Integrator
{
  virtual void
  integrate( double lowBound,
            double upBound,
            int count ) = 0;
};
```

Integrator.f90

```
interface
  function integrate( lowBound,
                    upBound,
                    count )
    real(kind(1.0D0)) :: lowBound, upBound
    integer :: count
  end function
end interface
```

18

Publish the Interface in SIDL

- Publish the interface
 - interfaces are published in SIDL (Scientific Interface Definition Language)
 - can't publish in native language because of language interoperability requirement
- Integrator example:

```
interface Integrator extends cca.Port
{
    double integrate(in double lowBound,
                    in double upBound,
                    in int count);
}
```

19

F90 Integrator Interface

```
MODULE Integrator

interface

    !
    ! Returns the result of the integration from lowBound to upBound.
    !
    ! lowBound - the beginning of the integration interval
    ! upBound - the end of the integration interval
    ! count - the number of integration points
    !
    function integrate(port, lowBound, upBound, count)
        use CCA
        type(CCAPort) :: port
        real(kind(1.0D0)) :: integrate, lowBound, upBound
        integer :: count
    end function integrate

end interface

END MODULE Integrator
```

20

F90 Program

```
program Driver
  use CCA
  use MonteCarloIntegrator
  type (CCAPort) :: port

  print *, "Integral = ", integrate(port, 0.0D0, 1.0D0, 1000)
end program
```

21

C++ Abstract Integrator Class

```
/**
 * This abstract class declares the Integrator interface.
 */

class Integrator : public virtual gov::cca::port
{
public:
  virtual ~Integrator() { }

  /**
   * Returns the result of the integration from lowBound to upBound.
   *
   * lowBound - the beginning of the integration interval
   * upBound - the end of the integration interval
   * count - the number of integration points
   */
  virtual double integrate(double lowBound, double upBound, int count) = 0;
};
```

22

C++ Object-Oriented Program

```
#include <iostream>
#include "MonteCarloIntegrator.h"

int main(int argc, char* argv[])
{
    MonteCarloIntegrator* integrator = new MonteCarloIntegrator();

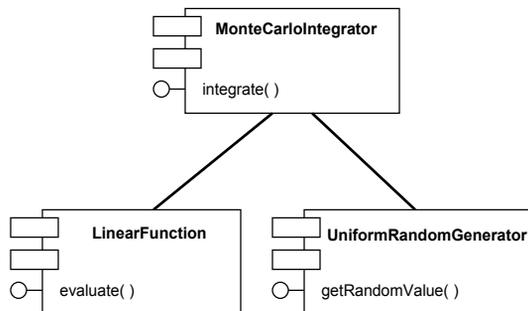
    cout << "Integral = " << integrator->integrate(0.0, 1.0, 1000) << endl;

    return 0;
}
```

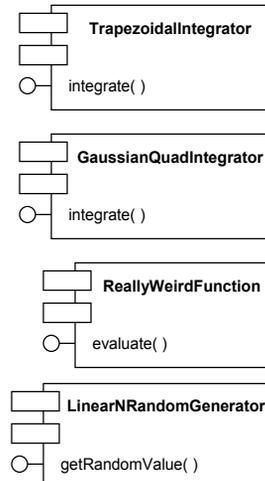
23

Component Program

Program



Component Library



24

Questions and Answers

- Is CCA similar to CORBA or COM/DCOM?
 - yes, but is a **component architecture** oriented towards high-performance computing
- Is CCA for parallel or distributed computing?
 - both, but currently only one or the other
- Can I use CCA today for scientific applications?
 - yes, but it is a research project
- Where can I get more information?
 - <http://www.cca-forum.org/>
 - join the CCA Forum

25

Final Thought

- Components are reusable assets. Compared with specific solutions to specific problems, components need to be carefully generalized to enable reuse in a variety of contexts. Solving a general problem rather than a specific one **takes more work**. In addition, because of the variety of deployment contexts, the creation of proper documentation, test suites, tutorials, online help texts, and so on is more demanding for components than for a specialized solution. *[Szyperski, p. 14]*

26

Bibliography

Booch, G. 1994. *Object-Oriented Analysis and Design with Applications*. Second Editions. Santa Clara, CA: The Benjamin/Cummings Publishing Company, p. 8.

Brooks, F. April 1987. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer* vol. 20(4), p. 12.

Cheesman, J. and J. Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*. New York, NY: Addison-Wesley

Courtois, P. June 1985. On Time and Space Decomposition of Complex Structures. *Communications of the ACM* vol 28(6), p. 596.

Gall, J. 1986. *Systemantics: How Systems Really Work and How They Fail*. Second Edition. Ann Arbor, MI: The General Systemantics Press, p. 65.

Simon, H. 1982. *The Sciences of the Artificial*. Cambridge, MA: The MIT Press, p. 217.

Szyperski, C. 1998. *Component Software: Beyond Object-Oriented Programming*. New York, NY: Addison-Wesley, p. 30

Next: **CCA Concepts**



Common Component Architecture Concepts

CCA Forum Tutorial Working Group

<http://www.cca-forum.org/tutorials/>
tutorial-wg@cca-forum.org



Goals

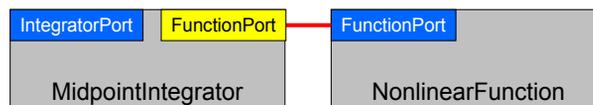
- Introduce essential features of the Common Component Architecture
- Provide common vocabulary for remainder of tutorial
- What distinguishes CCA from other component environments?

Features of the Common Component Architecture

- A component model specifically designed for high-performance computing
 - Support HPC languages (*Babel*)
 - Support parallel as well as distributed execution models
 - Minimize performance overhead
- Minimalist approach makes it easier to componentize existing software
- Component interactions are *not* merely dataflow
- Components are peers
 - No particular component assumes it is “in charge” of the others.
 - Allows the application developer to decide what is important.

3

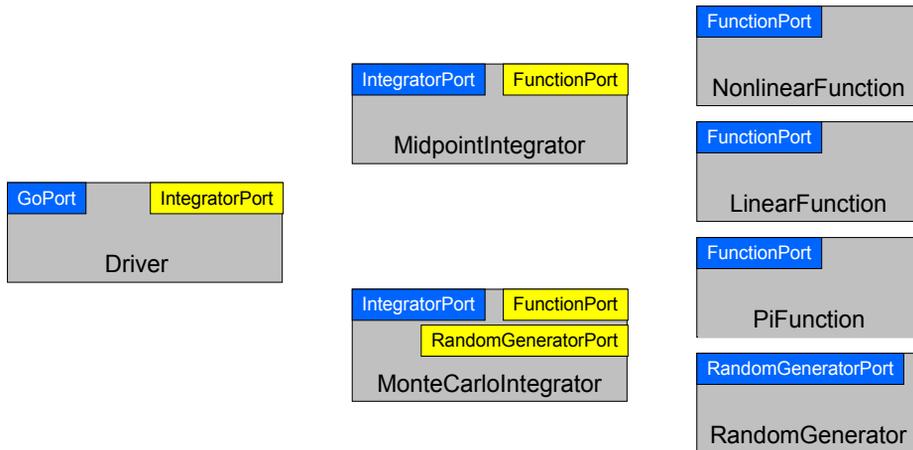
CCA Concepts: Ports



- Components interact through well-defined **interfaces**, or **ports**
 - In OO languages, a port is a class or interface
 - In Fortran, a port is a bunch of subroutines or a module
- Components may **provide** ports – **implement** the class or subroutines of the port
- Components may **use** ports – **call** methods or subroutines in the port
- Links denote a caller/callee relationship, **not dataflow!**
 - e.g., FunctionPort could contain: `evaluate(in Arg, out Result)`

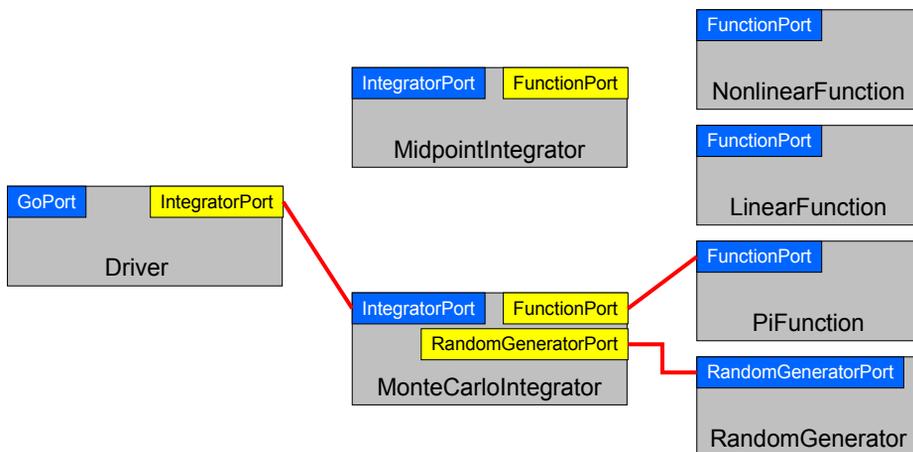
4

Components and Ports in the Integrator Example



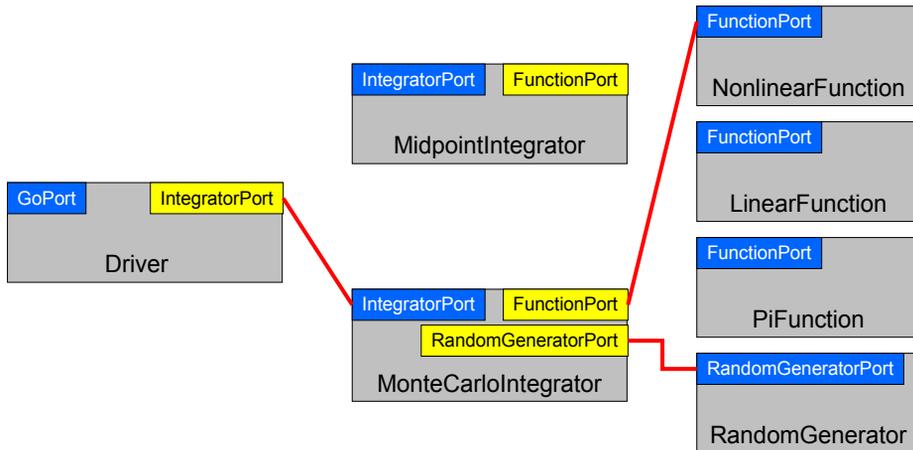
5

An Application Built from the Example Components

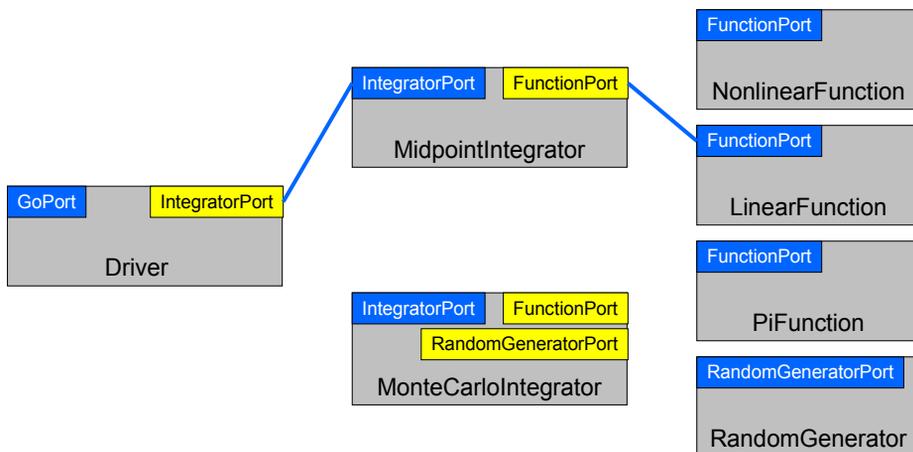


6

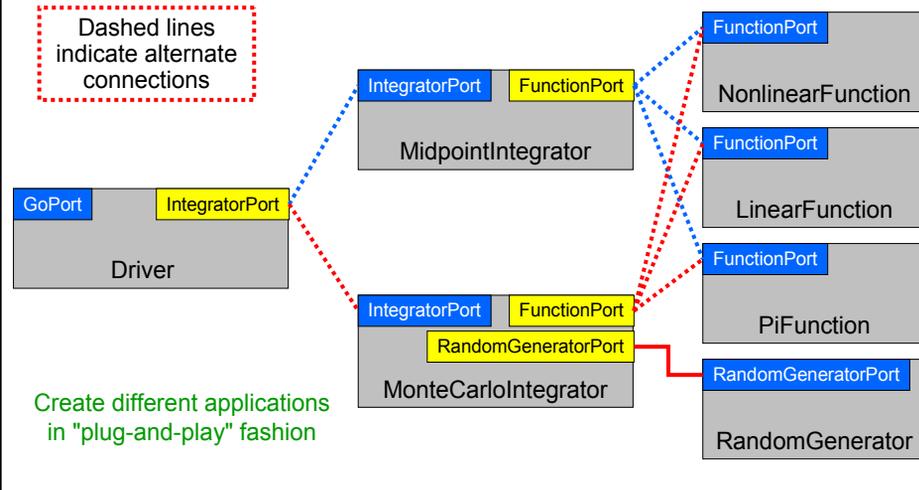
Another Application...



Application 3...



And Many More...

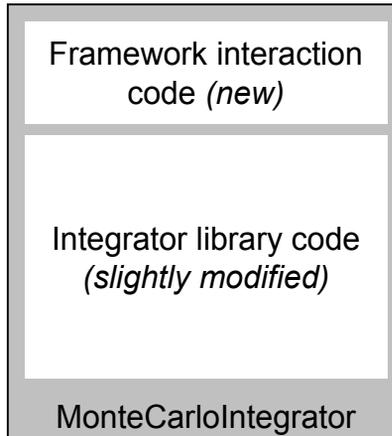


Ports, Interoperability, and Reuse

- Ports (interfaces) define how components interact
- Generality, quality, robustness of ports is up to designer/architect
 - “Any old” interface is easy to create, but...
 - Developing a robust domain “standard” interface requires thought, effort, and cooperation
- General “plug-and-play” interoperability of components requires multiple implementations conforming to the same interface
- Designing for interoperability and reuse requires “standard” interfaces
 - Typically domain-specific
 - “Standard” need not imply a formal process, may mean “widely used”

Components vs Libraries

- Component environments **rigorously** enforce interfaces
- Can have **several versions** of a component loaded into a single application
- Component needs add'l code to interact w/ framework
 - Constructor and destructor methods
 - Tell framework what ports it *uses* and *provides*
- Invoking methods on other components requires slight modification to “library” code



11

CCA Concepts: Frameworks

- The framework provides the means to “hold” components and **compose** them into applications
 - The framework is often application’s “main” or “program”
- Frameworks allow **exchange of ports** among components without exposing implementation details
- Frameworks provide a small set of **standard services** to components
 - BuilderServices allow programs to compose CCA apps
- Frameworks may make themselves appear as components in order to connect to components in other frameworks
- *Currently*: specific frameworks support specific computing models (parallel, distributed, etc.).
Future: full flexibility through integration or interoperation

12

The Lifecycle of a Component

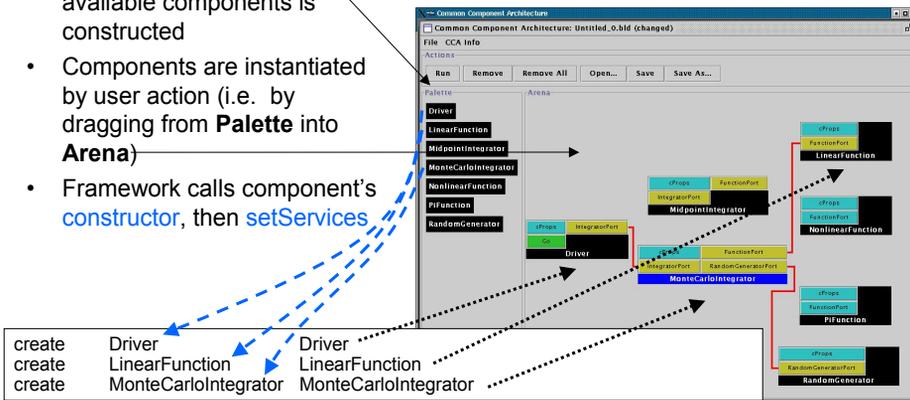
- User instructs framework to load and *instantiate* components
- User instructs framework to connect *uses* ports to *provides* ports
- Code in components uses functions provided by another component
- Ports may be disconnected
- Component may be destroyed

Look at actual code in next tutorial module

13

Loading and Instantiating Components

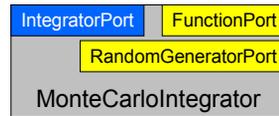
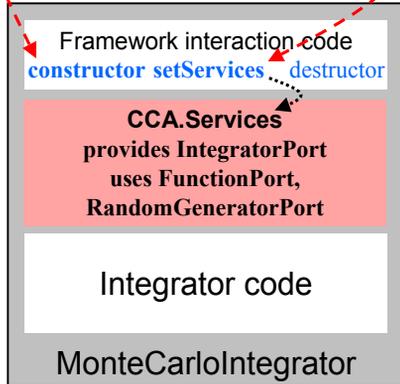
- Components are code (usu. library or shared object) + metadata
- Using metadata, a **Palette** of available components is constructed
- Components are instantiated by user action (i.e. by dragging from **Palette** into **Arena**)
- Framework calls component's **constructor**, then **setServices**
- Details are **framework-specific!**
- **Ccaffeine** currently provides both command line and GUI approaches



14

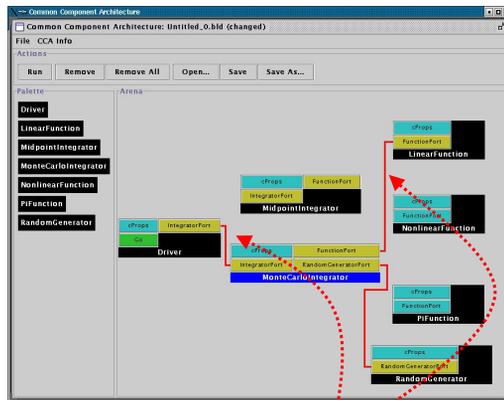
Component's View of Instantiation

- 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

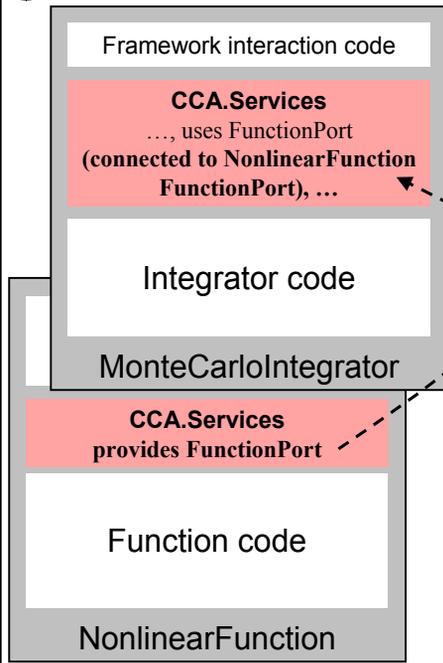


User Connects Ports

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



connect	Driver	IntegratorPort	MonteCarloIntegrator	IntegratorPort
connect	MonteCarloIntegrator	FunctionPort	LinearFunction	FunctionPort
...				

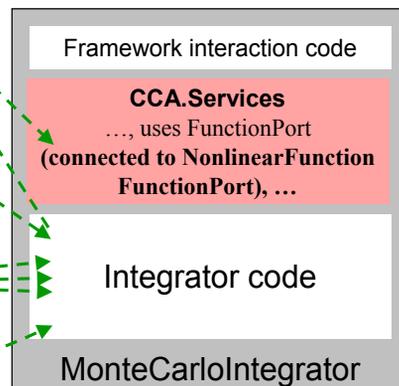


Component's View of Connection

- 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

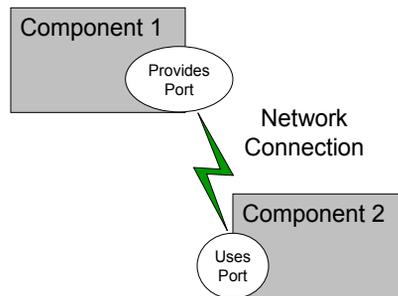
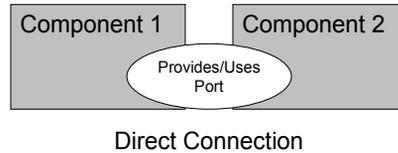
Component's View of Using a Port

- 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



Importance of Provides/Uses Pattern for Ports

- Fences between components
 - Components must **declare** both what they provide and what they use
 - Components **cannot interact** until ports are connected
 - No mechanism to call anything not part of a port
- Ports preserve high performance **direct connection** semantics...
- ...While also allowing **distributed computing**



19

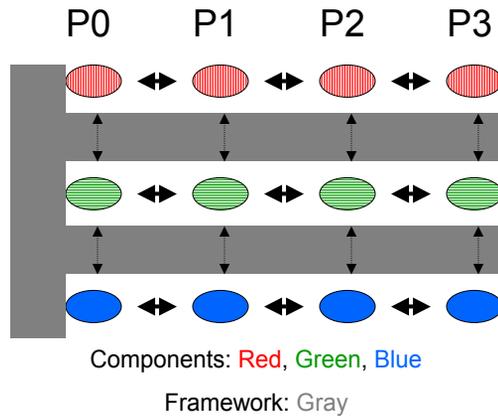
CCA Concepts: Direct Connection

- Components loaded into **separate namespaces** in the **same address space** (process) from shared libraries
- **getPort** call returns a pointer to the port's function table
- Calls between components equivalent to a C++ **virtual function call**: lookup function location, invoke
- Cost equivalent of ~2.8 F77 or C function calls
- All this happens "automatically" – **user just sees high performance**
- *Description reflects **Ccaffeine** implementation, but similar or identical mechanisms in other direct connect fwks*

20

CCA Concepts: Parallel Components

- **Single component multiple data** (SCMD) model is component analog of widely used SPMD model
- Each process loaded with the same set of components wired the same way
- **Different components** in **same process** “talk to each” other via ports and the framework
- **Same component** in **different processes** talk to each other through their favorite communications layer (i.e., MPI, PVM, GA)
- Also supports **MPMD/MCMD**

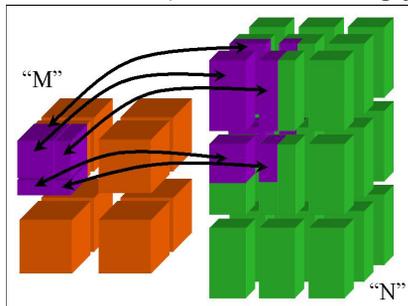


*Framework stays “out of the way”
of component parallelism*

21

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)

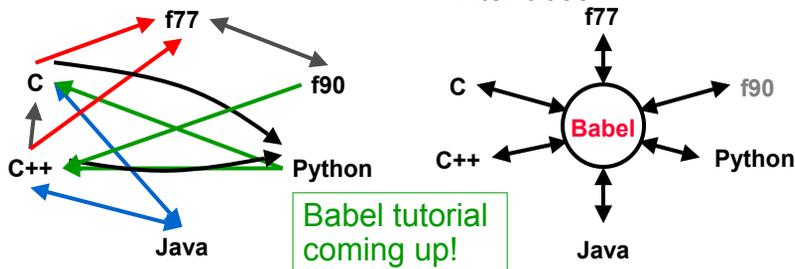


Research area -- tools under development

22

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



23

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)

24

Next: A Simple CCA Example



A Simple CCA Component Application

CCA Forum Tutorial Working Group

<http://www.cca-forum.org/tutorials/>
tutorial-wg@cca-forum.org



Module Overview

- What the example does: the math.
- From math to components: the architecture.
- The making of components: inheritance and ports.
- Framework-component interactions.
- Putting it all together: the CCafeine ways.
- The application in action.

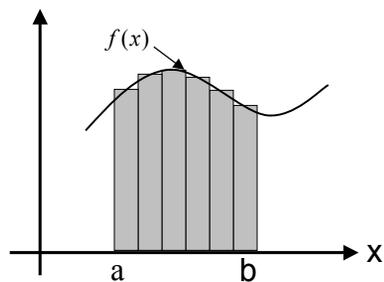
Goals

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

The Math: Integrator (1)

The midpoint numerical integrator

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

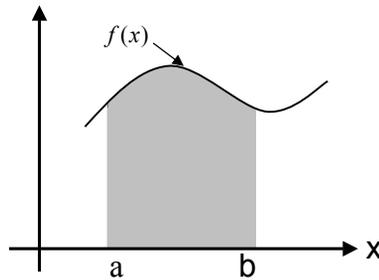


The Math: Integrator (2)

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 Uniformly distributed in $[a, b]$



5

The math: Functions

Linear Function

$$f_1(x) = 2x$$

Nonlinear Function

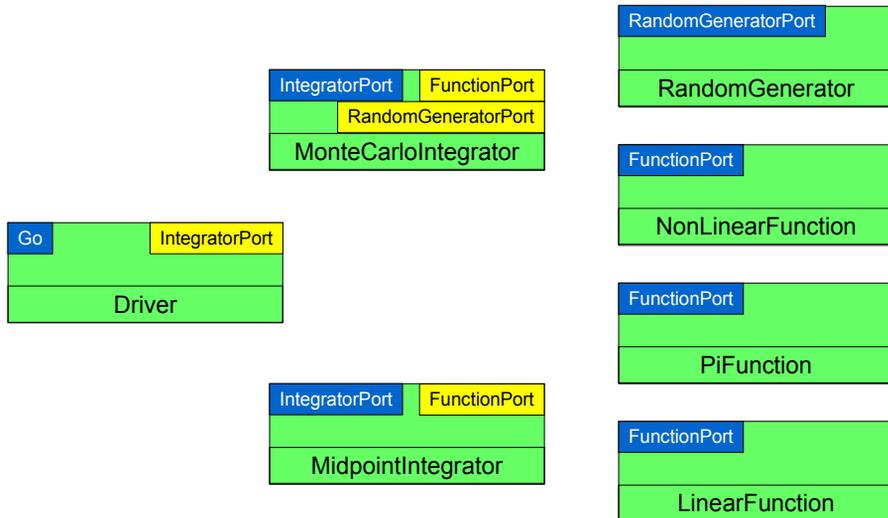
$$f_2(x) = x^2$$

Pi Function

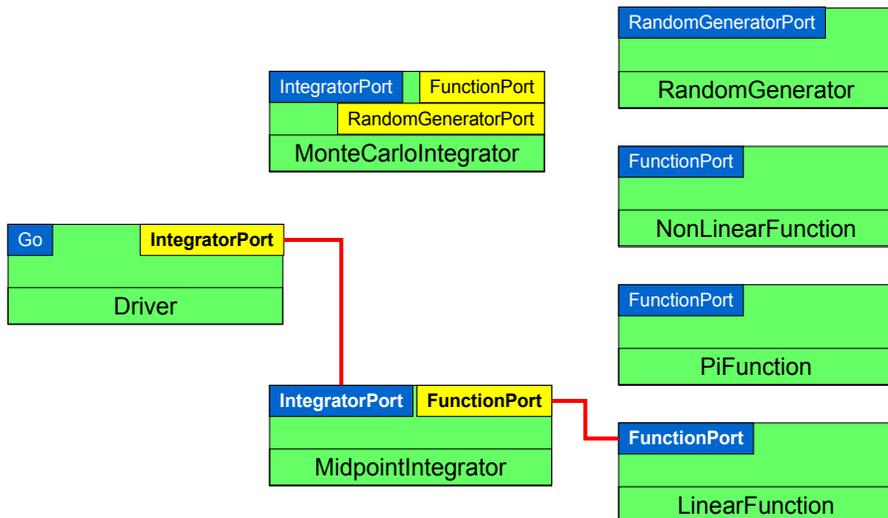
$$f_3(x) = \frac{4}{1+x^2}$$

6

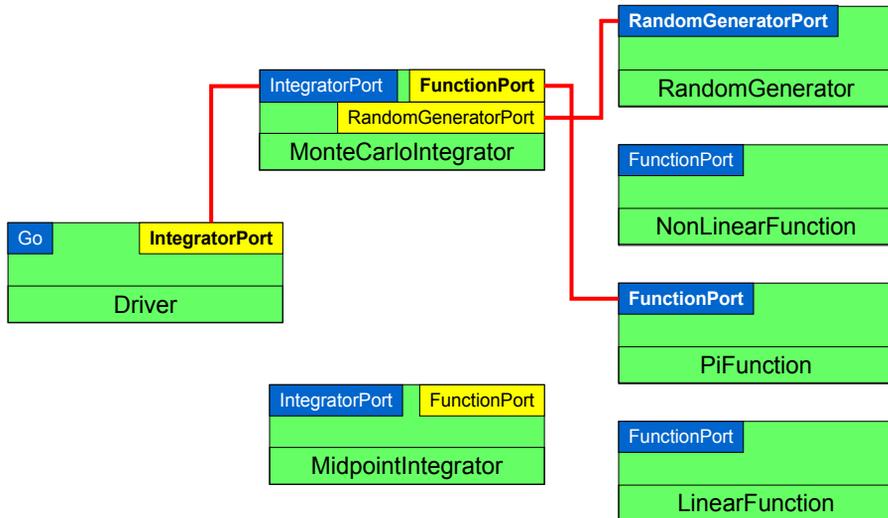
Available Components



Pluggability: Scenario 1

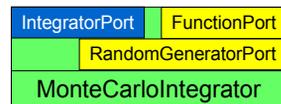
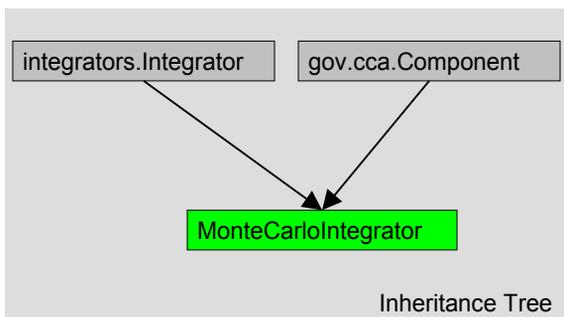


Pluggability: Scenario 2



9

MonteCarloIntegrator in Details



Relevant files:
 integrator.sidl
 function.sidl
 random.sidl

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

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

10

Saying it in SIDL

```
version integrators 1.0;

package integrators {

    interface Integrator
        extends gov.cca.Port
    {
        double integrate(in double lowBound,
            in double upBound, in int count);
    }
    class MonteCarloIntegrator
        implements-all Integrator,
            gov.cca.Component
    {
    }
    .....
}
```

11

Notes

- Inheritance from ***gov.cca.Component*** furnishes the only method known to the framework: ***setServices()***
- “**Provides**” ports are interfaces that need to inherit from ***gov.cca.Port*** (***Integrator*** in this case)

12

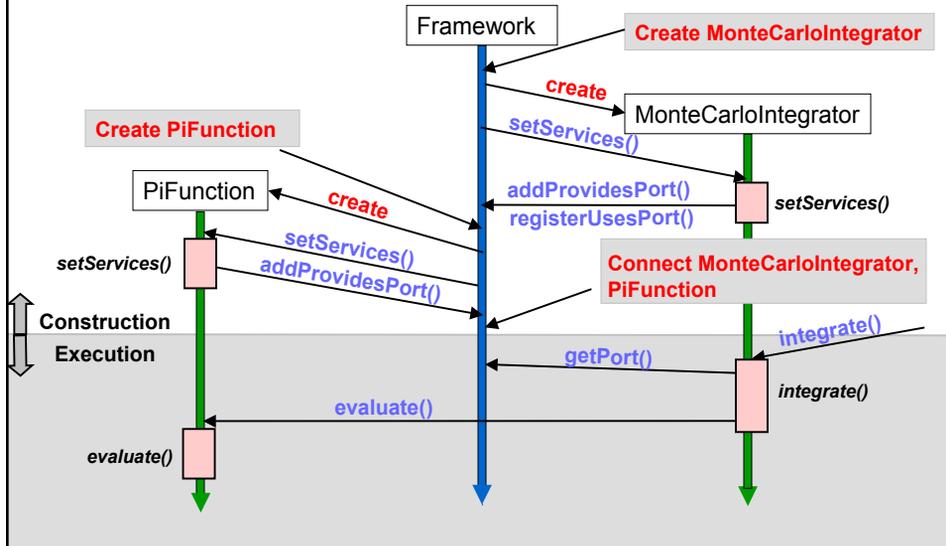
The Framework Role

- Framework-to-Component: **setServices()**
 - Called after the component is constructed.
 - The component's chance to identify:
 - Ports it provides – **addProvidesPort()**
 - Ports it uses – **registerUsesPort()**
 - Component should not acquire the port here – Reason: it may not be there yet !!!!
 - Also used to “shutdown” the component.

Component-to-Framework

- Mainly through **Services** object passed through **setServices()**.
- **addProvidesPort(), registerUsesPort():**
 - Component “pointer”, PortName, PortType, PortProperties.
- **getPort():**
 - Called by the using component.
 - Matching using portType (not name).
- **releasePort(), removeProvidesPort():**
 - When all is done.

The Life Cycle Revisited



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,
        portType ← "IntegratorPort",
        "integrators.Integrator", tm);
    // The Ports I use
    frameworkServices.registerUsesPort (
        "FunctionPort",
        "functions.Function", tm);
    frameworkServices.registerUsesPort (
        "RandomGeneratorPort",
        "randomgen.RandomGenerator", tm);
}
.....
    
```

Diagram annotations:

- `portType` points to `"IntegratorPort"`.
- `portName` points to `"IntegratorPort"`.
- `portProperties` points to `"integrators.Integrator"`.

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.

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

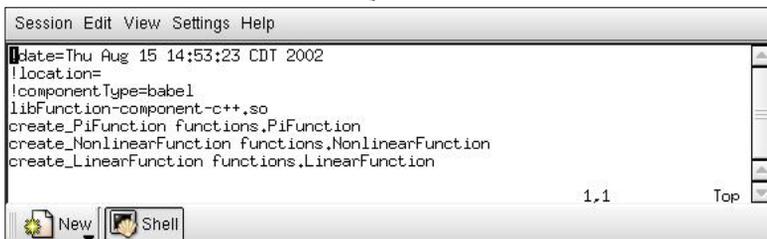
```
.....  
functions::Function functionPort;  
randomgen::RandomGenerator randomPort;  
double sum = 0.0;  
randomPort = frameworkServices.getPort ("RandomGeneratorPort");  
functionPort = frameworkServices.getPort ("FunctionPort");  
for (int i = 0; i < count; i++){  
    double x = lowBound + (upBound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

Putting it all together

- Getting the application to do something:
 - Assembling the components into an app.
 - Launching the Application.
- App 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.
- App 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.

Oh Component , where art thou?.

Which components, and how to create them



The screenshot shows a terminal window with the following content:

```

Session Edit View Settings Help
!date=Thu Aug 15 14:53:23 CDT 2002
!location=
!componentType=babel
libFunction=component-c++.so
create_PiFunction functions.PiFunction
create_NonLinearFunction functions.NonlinearFunction
create_LinearFunction functions.LinearFunction
  
```

At the bottom of the terminal window, there are buttons for 'New' and 'Shell', and the text '1,1 Top' is visible on the right side.

More details in the Ccaffeine Module

App. Assembly The Ccaffeine way

Session Edit View Settings Help

```

repository get functions.PiFunction
repository get integrators.MonteCarloIntegrator
repository get integrators.MidpointIntegrator
repository get integrators.ParallelIntegrator
repository get tutorial.Driver

# Instantiate and name components that have been loaded
create randomgen.RandRandoGenerator rand
# f(x) = 4.0 / (1 + x^2)
create functions.PiFunction function
create integrators.MonteCarloIntegrator integrator
create tutorial.Driver driver

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

# Good to Go!
go driver GoPort

bye
          
```

Command line "script"

Common Component Architecture: Untitled.0.bld (changed)

File CCA Info

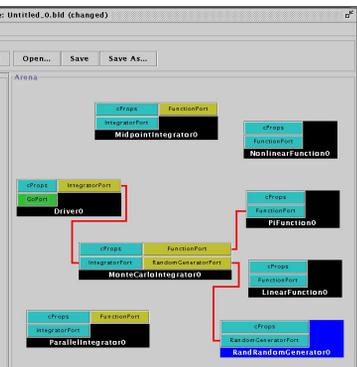
Actions

Run Remove Remove All Open... Save Save As...

Palette

- LinearFunction
- NonlinearFunction
- PiFunction
- MidpointIntegrator
- MonteCarloIntegrator
- ParallelIntegrator
- RandRandoGenerator
- Driver

Arena



GUI Interface

21


CCA
 Common Component Architecture

A Simple CCA Component Application

Next: Babel

22



Language Interoperable CCA Components via

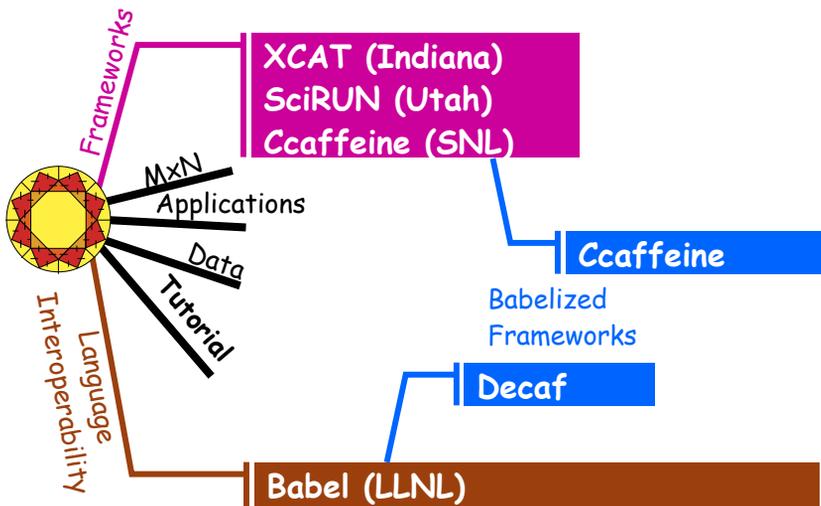


CCA Forum Tutorial Working Group

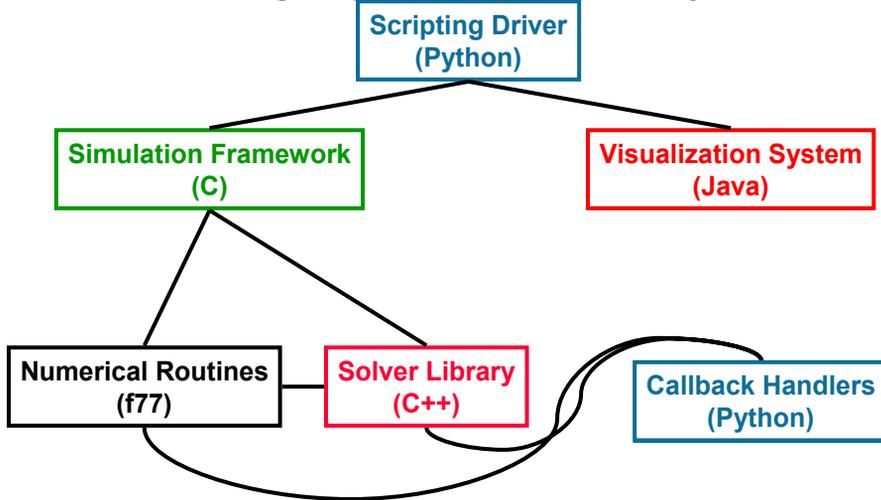
<http://www.cca-forum.org/tutorials/>
tutorial-wg@cca-forum.org



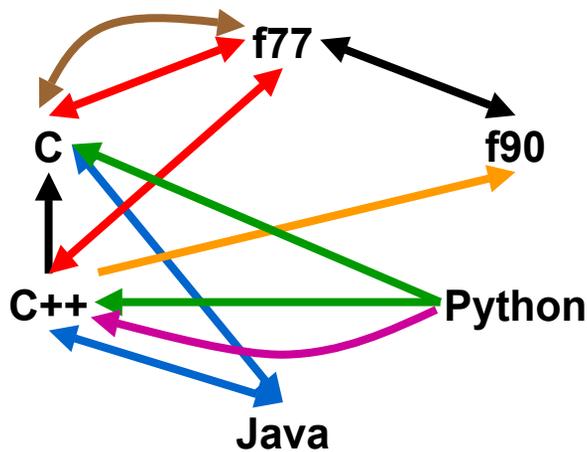
History of Babel & CCA



What I mean by “Language Interoperability”

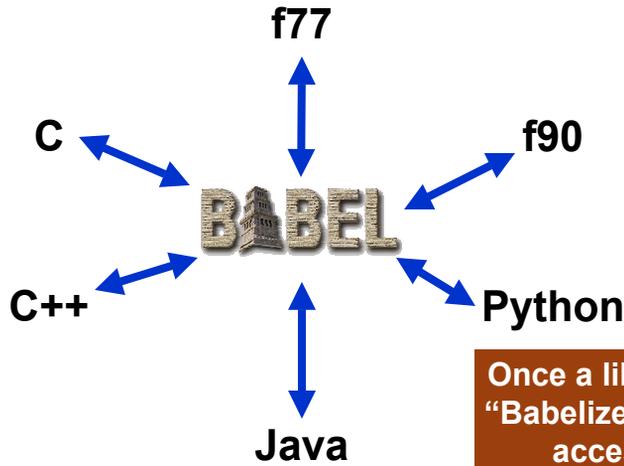


One reason why mixing languages is hard



- Native
- cfortran.h
- SWIG
- JNI
- Siloon
- Chasm
- Platform
Dependent

Babel makes all supported languages peers



This is not an LCD Solution!

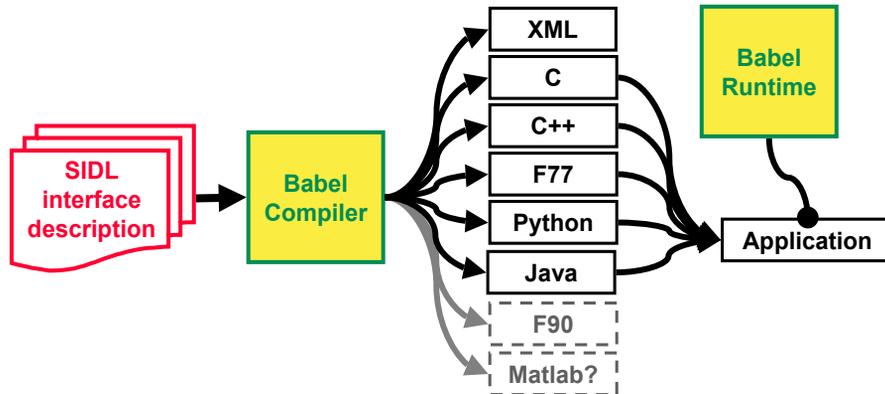
Once a library has been "Babelized" it is equally accessible from all supported languages

Babel Module's Outline

- Introduction
- ➔ Babel Basics
 - What Babel does and how
 - How to use Babel
 - Concepts needed for future modules
- Babel & CCA
 - Decaf Framework
 - Building language independent CCA components
 - Demo

Babel's Mechanism for Mixing Languages

- Code Generator
- Runtime Library

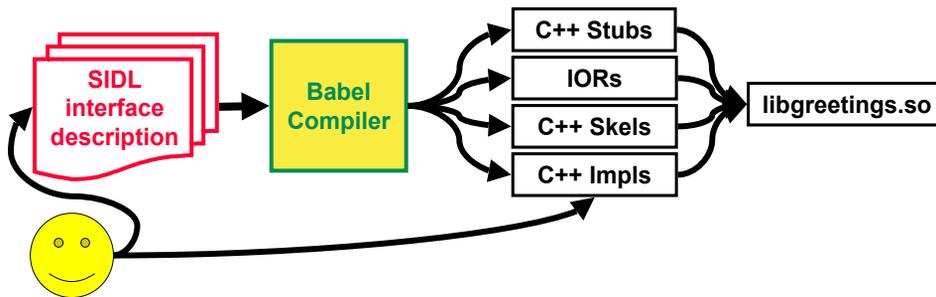


greetings.sidl: A Sample SIDL File

```

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

Library Developer Does This...



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

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)

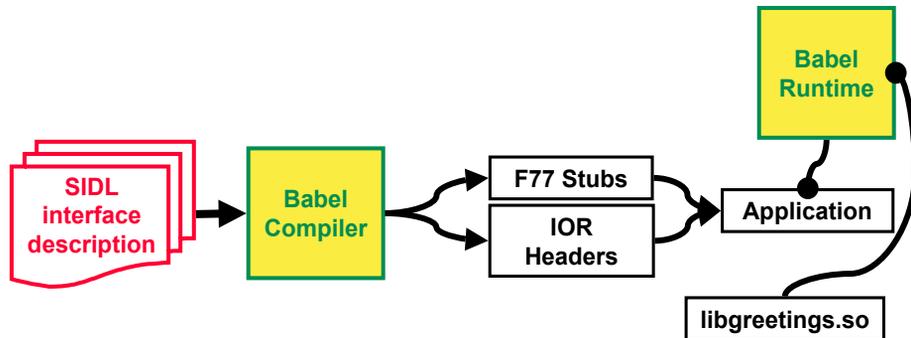
```

```

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...



- `babel --client=F77 greetings.sidl`
- Compile & Link generated Code & Runtime
- Place DLL in suitable location

SIDL 101: Classes & Interfaces

- SIDL has 3 user-defined objects
 - **Interfaces** – APIs only, no implementation
 - **Abstract Classes** – 1+ 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 101: 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)
 - Type: one of (bool, char, int, long, float, double, fcomplex, dcomplex, array<Type,Dimension>, enum, interface, class)
 - Name:

Babel Module's Outline

- Introduction
- Babel Basics
 - What Babel does and how
 - How to use Babel
 - Concepts needed for future modules
-  Babel & CCA
 - History & Current directions
 - Decaf Framework
 - Building language independent CCA components
 - Demo

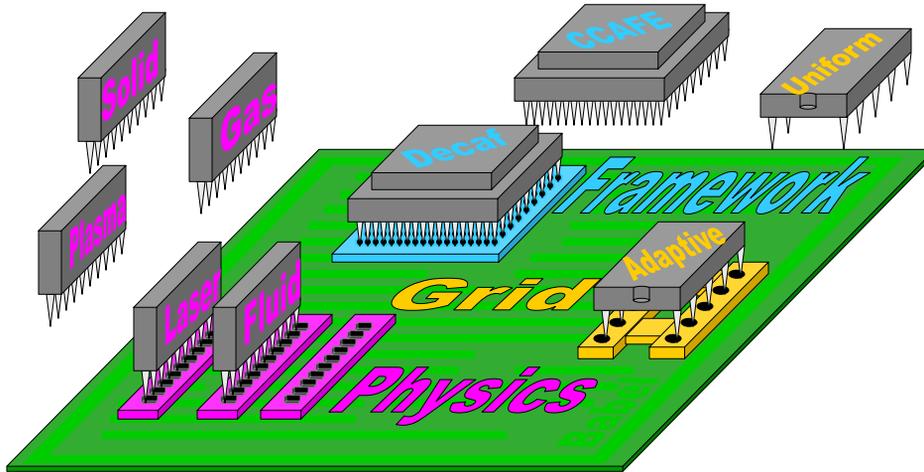
Decaf Details & Disclaimers

- Babel is a hardened tool
- Decaf is an example, not a product
 - Demonstrate Babel’s readiness for “real” CCA frameworks
 - Maintained as a stopgap
 - Distributed in “examples” subdirectory of Babel
- Decaf has no GUI

The CCA Spec is a SIDL File

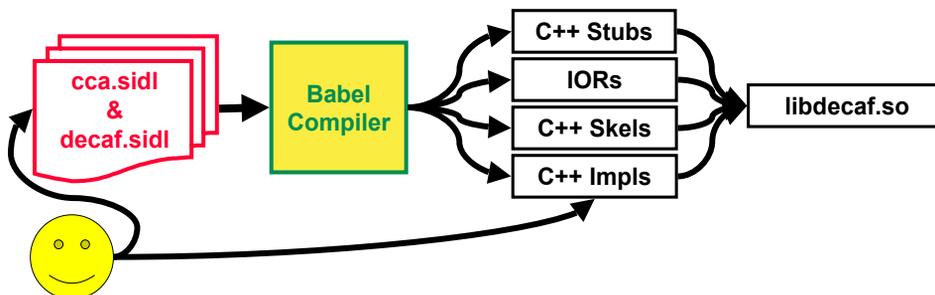
```
version gov.cca 0.6;
package gov {
package cca {
    interface Port { }
    interface Component {
        void setServices( in Services svcs );
    }
    interface Services {
        Port getPort( in string portName );
        registerUsesPort( /*etc*/ );
        addProvidesPort( /*etc*/ );
    }
}
/*etc*/
```

The CCA from Babel's POV



17

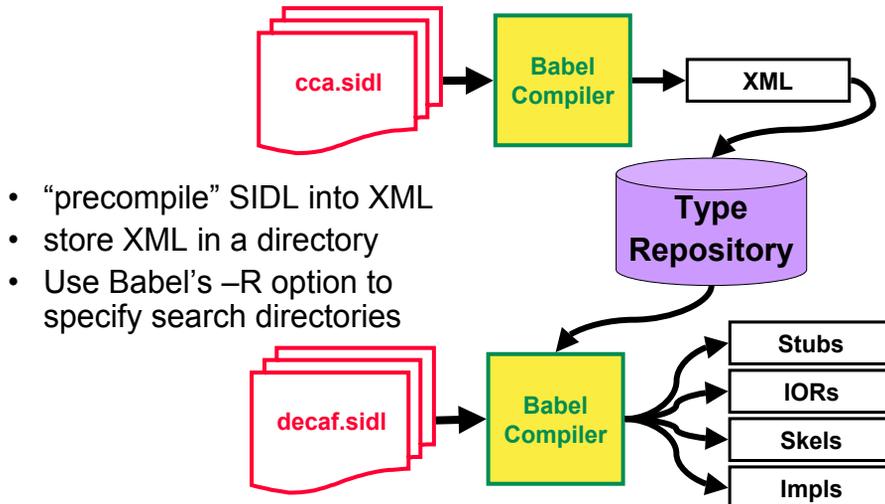
How I Implemented Decaf



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

18

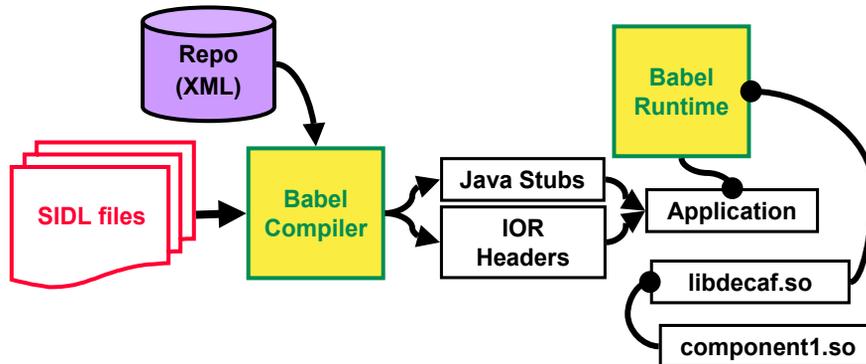
An Extra Babel Tip



How to Use CCA Components and Decaf

- Decaf doesn’t provide a GUI
- Simply program by explicitly
 - creating components
 - connecting ports
 - invoking the “goPort”
- Use Babel as needed to generate bindings in your language of choice
- Make sure Babel Runtime can locate DLLs for Decaf and any CCA components.

To Use the Decaf Framework



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

Example: A Driver in Python

```

import decaf.Framework
import gov.cca.ports.GoPort
if __name__ == '__main__':
    fwk = decaf.Framework.Framework()

    server = fwk.createInstance( "ServerName",
                                "HelloServer.Component", 0 )
    client = fwk.createInstance( "ClientName",
                                 "HelloClient.Component", 0 )

    fwk.connect(server,"HelloPort",
                client,"HelloPort" )

    port = fwk.lookupPort(client,"GoPort")
    go = gov.cca.ports.GoPort.GoPort( port )
    go.go()
  
```

How to Write and Use Babelized CCA Components

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

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

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

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

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

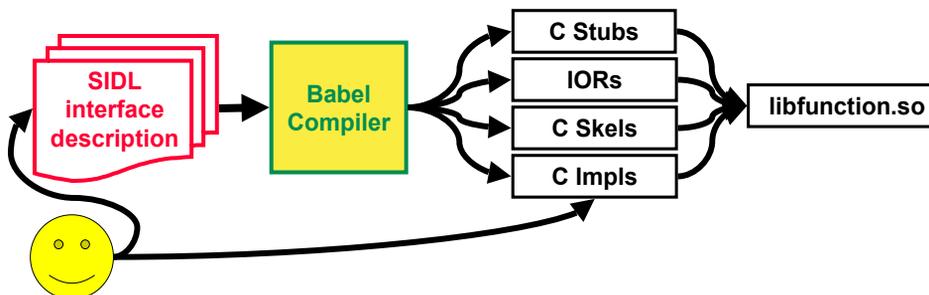
- Define “Components” that implement those Ports
 - CCA Component =
 - SIDL Class
 - implements gov.cca.Component (& any provided ports)

```
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 { }
```

25

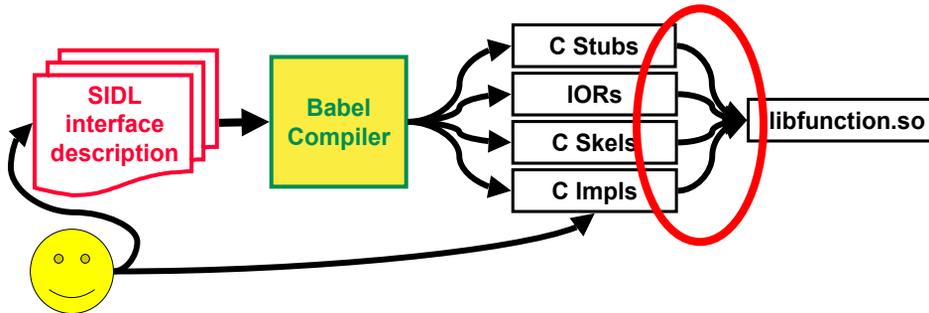
How to Write A Babelized CCA Component (3/3)



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

26

What's the Hardest Part of this Process?

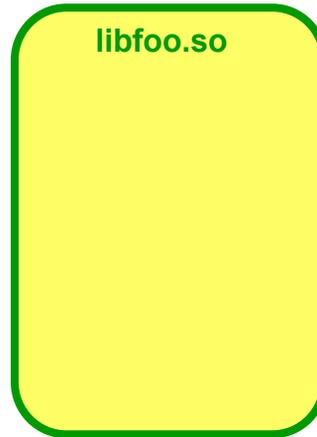


- Properly building dynamically loadable .so files.

Review of “Linkage”

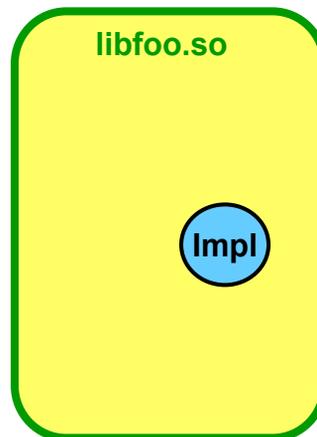
- Static Linked Libraries (*.a)
 - Symbols are hardcoded
 - Resolved at link-time of application
- Shared Object Libraries (*.so)
 - Symbols are hardcoded
 - Symbols resolved at load time (before main())
- Dynamically Loaded Libraries (*.so) (*.dll in Win32)
 - Symbols are determined at run time (by app code)
 - Symbols resolved at run time (void* dlopen(char*))

What goes into a DLL?



What goes into a DLL?

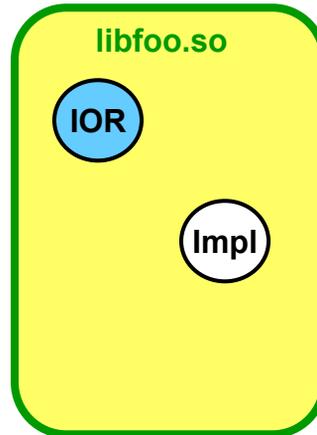
1. The Type's Impl
 - Where all the guts of the component lives.



What goes into a DLL?

2. The Type's IOR

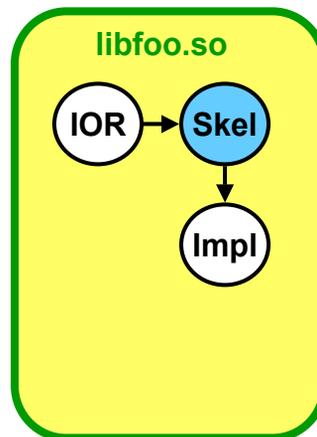
- IORs (Intermediate Object Representation)
- Always implemented in ANSI C
- Babel Object Model is implemented in IOR
- Dynamic Loading is based on symbols in IOR



What goes into a DLL?

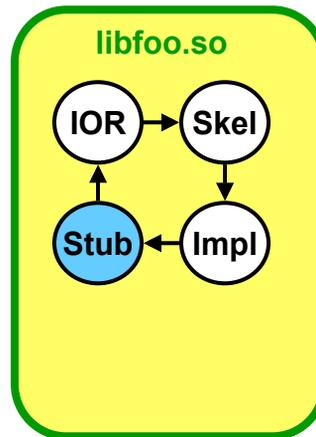
3. The Type's Skel

- IORs depend on the Skels
- Skels translate from ANSI C to Impl language
- Skels call Impls



What goes into a DLL?

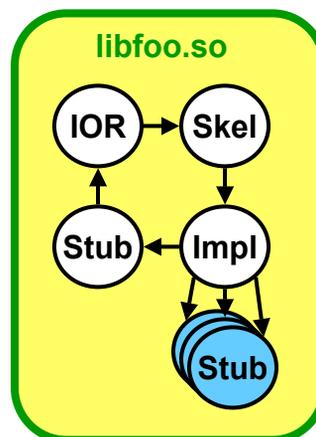
4. The Type's Stub
- Impl depends on Stubs
 - class needs to call methods on itself
 - Like "this" pointer in C++
 - self in Python
 - Stubs translate from application Language to ANSI C



33

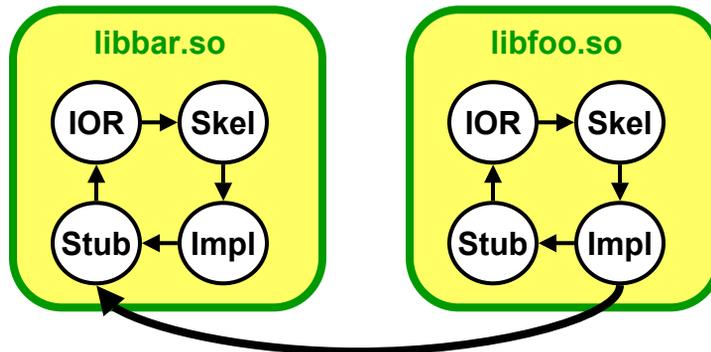
What goes into a DLL?

5. Stubs for all the other types that are
- passed as arguments,
 - return values, or
 - manipulated internally in the Type's Impl



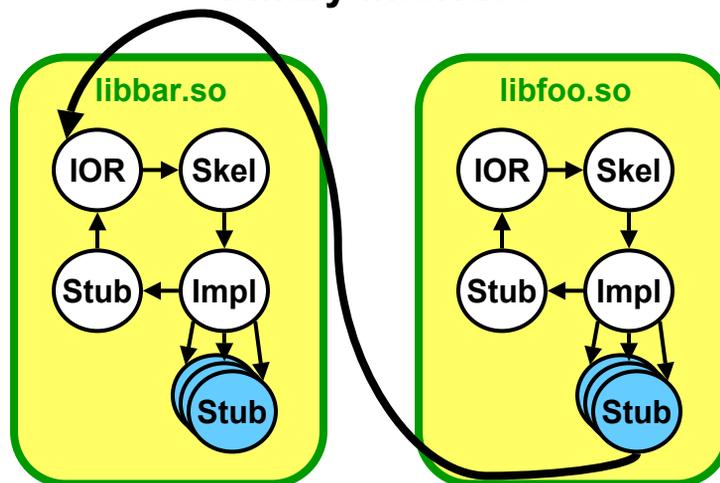
34

Q: Why not keep each Stub exclusively with its own Impl?

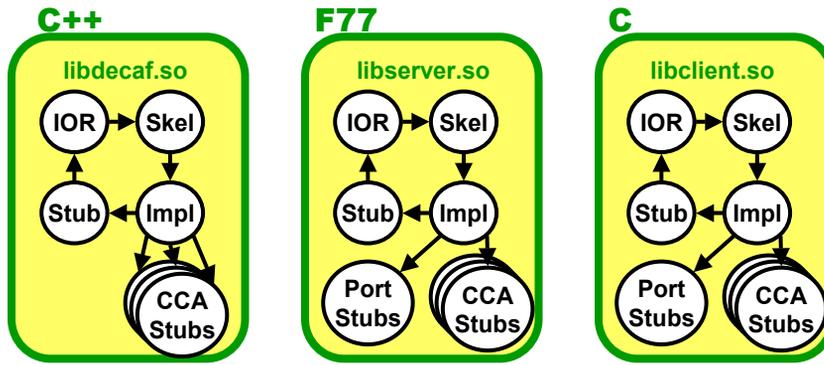


A: Works only if bar_Impl and foo_Impl are implemented in the same language

IORs provide a language-independent binary interface



What you'll see with the upcoming "Hello World" demo



And a "main" in any of



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]

UCRL-PRES-148796

5, July 2002



This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48



Writing Components

CCA Forum Tutorial Working Group

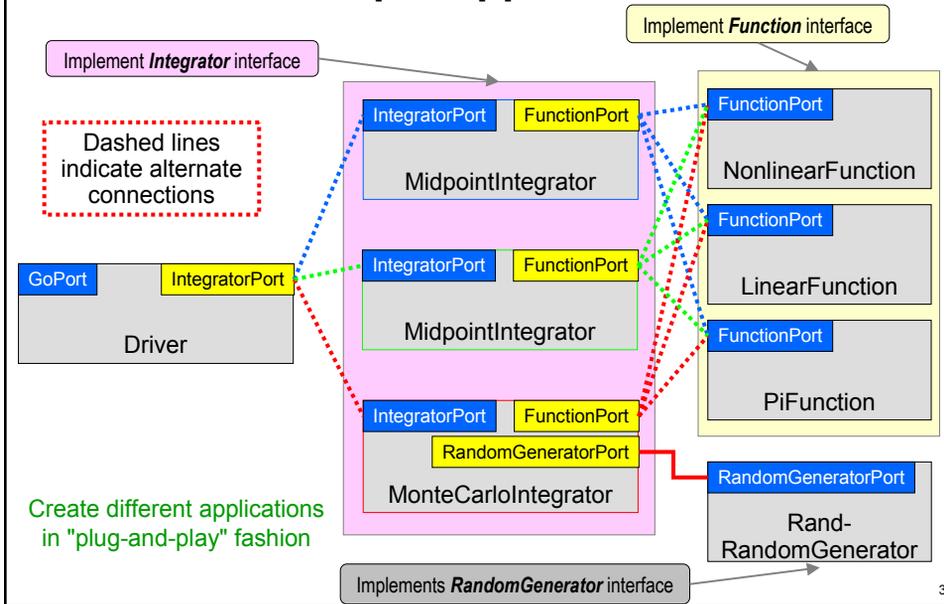
<http://www.cca-forum.org/tutorials/>
tutorial-wg@cca-forum.org



Module Overview

- Goal: present a step-by-step approach to creating CCA components
- Example application
- Steps involved in writing CCA components
 1. Interface definition; ports
 2. Component implementation
 1. Framework interactions
 2. Component interactions: uses and provides ports
 3. Compiling
 4. Running

Example Applications



Interface Definition

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

MonteCarloIntegrator Component

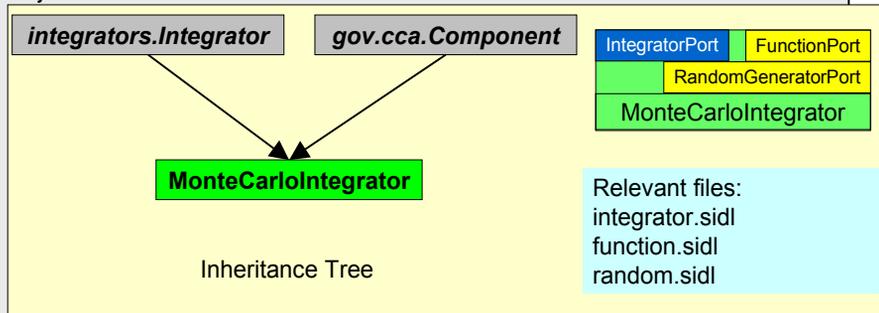
1. Use Babel to generate C++ skeletons and implementation files from integrator.sidl
2. Fill in implementation details in integrator-component-c++/:
 - integrator_MonteCarloIntegrator_Impl.hh
 - integrator_MonteCarloIntegrator_Impl.cc
3. Create C wrapper functions (for component creation):
 - integrator_Integrator_wrapper_Impl.cc
4. Create makefile and build dynamic library
 - Makefile
 - libIntegrator-component-c++.so
5. Create integrator.cca (Ccaffeine-specific)

Integrator Port

```
version integrators 1.0;

package integrators {

  interface Integrator extends gov.cca.Port {
    double integrate(in double lowBound,
                    in double upBound, in int count);
  }
}
```



Using Babel to Create The Repository

- A repository containing XML versions of the SIDL definition is created first; it will be used for name resolution later
- Makefile fragment (for all SIDL definitions in this example):

```
SIDLFILES = cca.sidl integrator.sidl function.sidl \  
            random.sidl driver.sidl  
  
.repository: $(SIDLFILES)  
    rm -f repository/*.xml \  
    babel --xml --repository-path=repository \  
    --output-directory=repository $(SIDLFILES)  
    touch .repository
```

7

Using Babel to Generate Code

- Makefile fragment (top-level directory):

```
.integrator-component-c++: integrator.sidl cca.sidl  
    babel --server=C++ --repository-path=repository \  
    --output-directory=integrator-component-c++ \  
    --suppress-timestamp integrators \  
    randomgen.RandomGenerator functions.Function  
    touch .integrator-component-c++
```

- Important: the *randomgen.RandomGenerator* and *functions.Function* interfaces are referenced by the Integrator implementation(s) and are thus included in the command line for generating the sources for the integrators package.

8

Contents of integrator-component-c++/

SIDL.hh	gov_cca_ComponentID_IOR.c	integrators_Integrator.hh
SIDL_BaseClass.cc	gov_cca_ComponentID_IOR.h	integrators_Integrator_IOR.c
SIDL_BaseClass.hh	gov_cca_Component_IOR.c	integrators_Integrator_IOR.h
SIDL_BaseException.cc	gov_cca_Component_IOR.h	integrators_Integrator_wrapper_Impl.cc
SIDL_BaseException.hh	gov_cca_Port.cc	integrators_MidpointIntegrator.cc
SIDL_BaseInterface.cc	gov_cca_Port.hh	integrators_MidpointIntegrator.hh
SIDL_BaseInterface.hh	gov_cca_Port_IOR.c	integrators_MidpointIntegrator_IOR.c
SIDL_DLL.cc	gov_cca_Port_IOR.h	integrators_MidpointIntegrator_IOR.h
SIDL_DLL.hh	gov_cca_Services.cc	integrators_MidpointIntegrator_Impl.cc
SIDL_Loader.cc	gov_cca_Services.hh	integrators_MidpointIntegrator_Impl.hh
SIDL_Loader.hh	gov_cca_Services_IOR.c	integrators_MidpointIntegrator_Skel.cc
babel.make	gov_cca_Services_IOR.h	integrators_MonteCarloIntegrator.cc
functions_Function.cc	gov_cca_Type.hh	integrators_MonteCarloIntegrator.hh
functions_Function.hh	gov_cca_TypeMap.cc	integrators_MonteCarloIntegrator_IOR.c
functions_Function_IOR.c	gov_cca_TypeMap.hh	integrators_MonteCarloIntegrator_IOR.h
functions_Function_IOR.h	gov_cca_TypeMap_IOR.c	integrators_MonteCarloIntegrator_Impl.cc
gov_cca_CCAException.cc	gov_cca_TypeMap_IOR.h	integrators_MonteCarloIntegrator_Impl.hh
gov_cca_CCAException.hh	gov_cca_TypeMismatchException.cc	integrators_MonteCarloIntegrator_Skel.cc
gov_cca_CCAExceptionType.hh	gov_cca_TypeMismatchException.hh	integrators_ParallelIntegrator.cc
gov_cca_CCAExceptionType_IOR.c	gov_cca_TypeMismatchException_IOR.c	integrators_ParallelIntegrator.hh
gov_cca_CCAExceptionType_IOR.h	gov_cca_TypeMismatchException_IOR.h	integrators_ParallelIntegrator_IOR.c
gov_cca_CCAException_IOR.c	gov_cca_TypeMismatchException_Impl.cc	integrators_ParallelIntegrator_IOR.h
gov_cca_CCAException_IOR.h	gov_cca_TypeMismatchException_Impl.hh	integrators_ParallelIntegrator_Impl.cc
gov_cca_CCAException_Impl.cc	gov_cca_TypeMismatchException_Skel.cc	integrators_ParallelIntegrator_Impl.hh
gov_cca_CCAException_Impl.hh	gov_cca_Type_IOR.c	integrators_ParallelIntegrator_Skel.cc
gov_cca_CCAException_Skel.cc	gov_cca_Type_IOR.h	randomgen_RandomGenerator.cc
gov_cca_Component.cc	integrators.cca	randomgen_RandomGenerator.hh
gov_cca_Component.hh	integrators.hh	randomgen_RandomGenerator_IOR.c
gov_cca_ComponentID.cc	integrators_IOR.h	randomgen_RandomGenerator_IOR.h
gov_cca_ComponentID.hh	integrators_Integrator.cc	



MonteCarloIntegrator Component (C++):Implementation Header

```

namespace integrators {

/**
 * Symbol "integrators.MonteCarloIntegrator" (version 1.0)
 */
class MonteCarloIntegrator_impl
{
private:
    // Pointer back to IOR.
    // Use this to dispatch back through IOR vtable.
    MonteCarloIntegrator self;

    // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator._implementation)
    // Put additional implementation details here...
    gov::cca::Services frameworkServices; ← Reference to framework Services object
    // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator._implementation)

    ...

}; // end class MonteCarloIntegrator_impl

} // end namespace integrators

```

MonteCarloIntegrator Component (C++): Framework Interaction

```

integrators::MonteCarloIntegrator_impl::setServices (
    /*in*/ gov::cca::Services services )
    throw ()
{
    // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.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);
        frameworkServices.registerUsesPort ("RandomGeneratorPort",
            "randomgen.RandomGenerator", tm);
    }
    // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.setServices)
}
    
```

Annotations for the code above:

- Save a pointer to the Services object (points to `frameworkServices = services;`)
- Port name (points to `"IntegratorPort"`)
- Port type (points to `"integrators.Integrator"`)
- TypeMap reference (points to `tm`)

MonteCarloIntegrator Component (C++): integrate() Method

```

double
integrators::MonteCarloIntegrator_impl::integrate (
    /*in*/ double lowBound, /*in*/ double upBound, /*in*/ int32_t count ) throw ()
{
    // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.integrate)
    gov::cca::Port port;
    double sum = 0.0;
    functions::Function function_m;
    randomgen::RandomGenerator random_m;

    random_m = frameworkServices.getPort ("RandomGeneratorPort");
    function_m = frameworkServices.getPort ("FunctionPort");

    for (int i = 0; i < count; i++) {
        double x = random_m.getRandomNumber ();
        sum = sum + function_m.evaluate (x);
    }

    frameworkServices.releasePort ("RandomGeneratorPort");
    frameworkServices.releasePort ("FunctionPort");

    return (upBound - lowBound) * sum / count;
    // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.integrate)
}
    
```

Annotations for the code above:

- Get a RandomGenerator reference (points to `random_m = frameworkServices.getPort ("RandomGeneratorPort");`)
- Get a Function reference (points to `function_m = frameworkServices.getPort ("FunctionPort");`)
- Get a random number (points to `double x = random_m.getRandomNumber ();`)
- Evaluate function at random value (points to `sum = sum + function_m.evaluate (x);`)
- Release ports (points to `frameworkServices.releasePort ("RandomGeneratorPort");` and `frameworkServices.releasePort ("FunctionPort");`)
- Return integral value (points to `return (upBound - lowBound) * sum / count;`)

MonteCarloIntegrator Component (F77): Framework Interaction

```

subroutine
& integrators_MonteCarloIntegrator_setServices_impl(
& services)
implicit none
integer*8 self
integer*8 services
C DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.setServices)
C Insert the implementation here...
integer*8 tm, excpt, retval, myport, myservices
common/MonteCarloState/myservices
myservices = services
call gov_cca_Services_addReference_f(services)
call gov_cca_Services_createTypeMap_f(myservices, tm, excpt)
call integrators_MonteCarloIntegrator_cast_f(self,
& Port type → 'gov.cca.Port', myport)
call gov_cca_Services_addProvidesPort_f(myservices, myPort,
& 'IntegratorPort', 'integrators.Integrator',
& tm, excpt)
C The ports I use
call gov_cca_Services_registerUsesPort_f(myservices,
& 'FunctionPort', 'functions.Function', tm, excpt)
call gov_cca_Services_registerUsesPort_f(myservices,
& 'RandomGeneratorPort',
& 'randomgen.RandomGenerator', tm, excpt)
C DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.setServices)
end
    
```

Save a handle to the Services object

TypeMap reference

Explicit cast to Port

Port type

Port name

TypeMap reference

Port name

MonteCarloIntegrator Component (F77): integrate() Method

```

subroutine integrators_MonteCarloIntegrator_integrate_impl(
& lowBound, upBound, count, retval)
implicit none
integer*8 self
double precision lowBound, upBound
integer*4 count
double precision retval
C DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.integrate)
C Insert the implementation here...
integer*8 excpt, port, funport, randport, myservices
common/MonteCarloState/myservices
double precision sum, h, x, temp
integer i
call gov_cca_Services_getPort_f(myservices, 'FunctionPort', port, excpt)
call gov_cca_Services_getPort_f(myservices, 'RandomGeneratorPort', port, excpt)
call gov_cca_Services_getPort_f(myservices, 'randomgen.RandomGenerator', randport)
sum = 0.0
do 50 i = 0, count, 1
call randomgen_RandomGenerator_getRandomNumber_f(randport, x)
x = lowBound + (upBound - lowBound) * x
call functions_Function_evaluate_f(funport, x, temp)
sum = sum + temp
50 end do
C Release ports
call gov_cca_Services_releasePort_f(myservices, 'FunctionPort', excpt)
call gov_cca_Services_releasePort_f(myservices, 'RandomGeneratorPort', excpt)
retval = (upBound - lowBound) * sum / count
C DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.integrate)
end
    
```

Get a Function reference

Get a RandomGenerator reference

Get a random number

Evaluate function at random value

Release ports

Return integral value

Writing the C Wrapper

- At present, Ccaffeine requires some C functions for dynamic loading of components; example for two components:

```

#include "integrators.hh"
#include "gov_cca_Component.hh"
#include <stdio.h>

extern "C" {
  gov::cca::Component create_MonteCarloIntegrator() {
    ::gov::cca::Component ex = integrators::MonteCarloIntegrator::_create();
    return ex;
  }
  gov::cca::Component create_ParallelIntegrator() {
    ::gov::cca::Component ex = integrators::ParallelIntegrator::_create();
    return ex;
  }
  char **getComponentList() {
    static char *list[3];
    list[0] = "create_MonteCarloIntegrator integrators.MonteCarloIntegrator";
    list[1] = "create_ParallelIntegrator integrators.ParallelIntegrator";
    list[2] = 0;
    return list;
  }
}
  
```

Annotations:

- Create a MonteCarloIntegrator instance (points to `integrators::MonteCarloIntegrator::_create()`)
- Create a ParallelIntegrator instance (points to `integrators::ParallelIntegrator::_create()`)
- Return a list of components contained in this dynamic library (points to `list`)
- C wrapper function name (points to `create_MonteCarloIntegrator`)
- Component name (points to `integrators.MonteCarloIntegrator`)

5

MonteCarloIntegrator: integrators.cca

- Ccaffeine-specific file specifying the name of the dynamic library and creation method for each component

```

!date=Thu Aug 15 14:53:23 CDT 2002
!location=
!componentType=babel
libIntegrator-component-c++.so
create_MonteCarloIntegrator integrators.MonteCarloIntegrator
create_ParallelIntegrator integrators.ParallelIntegrator
  
```

Annotations:

- Component type: "babel" or "classic" (C++) (points to `!componentType=babel`)
- C wrapper function name (points to `create_MonteCarloIntegrator`)
- Component name (points to `integrators.MonteCarloIntegrator`)

Note: This mechanism is expected to change soon

16

Parallel Example: **MidpointIntegrator** integrate() Method

```

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

    // Get Function port
    function_m = frameworkServices.getPort("FunctionPort"); ← Get a Function reference

    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); } ← Parallel environment details

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

Parallel Example: **MidpointIntegrator** integrate() Method (continued)

```

// ...
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_m.evaluate(x); ← Evaluate function
    }
    myresult = h * sum;

    MPI_Reduce(&myresult, &result, 1, MPI_DOUBLE, MPI_SUM, 0,
              MPI_COMM_WORLD); } ← Compute integral in parallel

    frameworkServices.releasePort("FunctionPort"); ← Release port
    printf("result is %f\n", result);
    return result; ← Return integral value

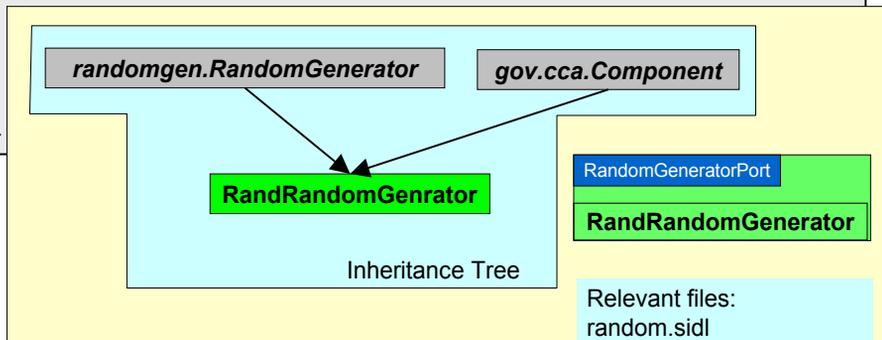
    // DO-NOT-DELETE splicer.end(integrators.ParallelIntegrator.integrate)
}
    
```

RandRandomGenerator Component

1. Use Babel to generate C++ skeletons and implementation files for random.sidl
2. Fill in implementation details in random-component-c++/:
 - randomgen_RandRandomGenerator_Impl.hh
 - randomgen_RandRandomGenerator_Impl.cc
3. Create C wrapper functions (for component creation).
 - randomgen_RandomGenerator_wrapper_Impl.cc
4. Create makefile and build dynamic library
 - Makefile
 - libRandom-component-c++.so
5. Create random.cca (Ccaffeine-specific)

RandomGenerator Port

```
version randomgen 1.0;
package randomgen {
  interface RandomGenerator extends gov.cca.Port {
    double getRandomNumber();
  }
}
```



RandRandomGenerator Component: Implementation Header

```

namespace randomgen {

/**
 * Symbol "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(randomgen.RandRandomGenerator._implementation)
    // Put additional implementation details here...
    gov::cca::Services frameworkServices; ← Reference to framework Services object
    // DO-NOT-DELETE splicer.end(randomgen.RandRandomGenerator._implementation)

    ...

}; // end class RandRandomGenerator_impl
} // end namespace randomgen
    
```

21

RandRandomGenerator Component: Framework Interaction

```

randomgen::RandRandomGenerator_impl::setServices (
    /*in*/ gov::cca::Services services )
throw ()
{
    // DO-NOT-DELETE splicer.begin(randomgen.RandRandomGenerator.setServices)
    frameworkServices = services; ← Save a pointer to the Services object
    if (frameworkServices._not_nil ()) {
        gov::cca::TypeMap tm = frameworkServices.createTypeMap ();

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

        // Port provided by RandomGenerator implementations
        frameworkServices.addProvidesPort (p, "RandomGeneratorPort",
            "randomgen.RandomGenerator", tm); ← Port name, TypeMap reference
        // No ports are used by this RandomGenerator implementation
    }
    // DO-NOT-DELETE splicer.end(randomgen.RandRandomGenerator.setServices)
}
    
```

22

PiFunction Component

1. Use Babel to generate C++ skeletons and implementation files for function.sidl
2. Fill in implementation details in function-component-c++/:
 - functions_PiFunction_Impl.hh
 - functions_PiFunction_Impl.cc
3. Create C wrapper functions (for component creation).
 - functions_Function_wrapper_Impl.cc
4. Create makefile and build dynamic library
 - Makefile
 - libFunction-component-c++.so
5. Create functions.cca (Ccaffeine-specific)

Function Port

```

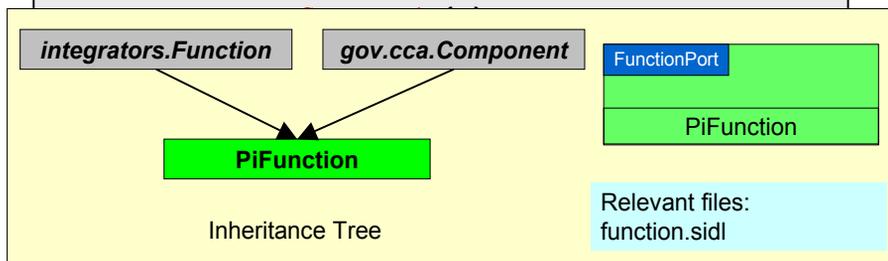
version functions 1.0;

package functions {

  interface Function extends gov.cca.Port {
    double evaluate(in double x);
  }

  class PiFunction implements-all Function,

```



PiFunction Component: Implementation Header

```

namespace functions {

/**
 * Symbol "function.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; ← Reference to framework Services object
  // DO-NOT-DELETE splicer.end(functions.PiFunction._implementation)

  ...

}; // end class PiFunction_impl
} // end namespace functions
  
```

25

PiFunction Component: Framework Interaction

```

functions::PiFunction_impl::setServices (
 /*in*/ gov::cca::Services services )
throw ()
{
  // DO-NOT-DELETE splicer.begin(functions.PiFunction.setServices)
  frameworkServices = services; ← Save a pointer to the Services object
  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, "FunctionPort",
    ← Port type → "functions.Function", tm); ← TypeMap reference
    ← Port name →
    // No Ports are used by this Function implementation
  }
  // DO-NOT-DELETE splicer.end(functions.PiFunction.setServices)
}
  
```

26

Driver Component

1. Use Babel to generate C++ skeletons and implementation files for driver.sidl
2. Fill in implementation details in driver-component-c++/:
 - tutorial_Driver_Impl.hh
 - tutorial_Driver_Impl.cc
3. Create C wrapper functions (for component creation).
 - tutorial_Driver_wrapper_Impl.cc
4. Create makefile and build dynamic library
 - Makefile
 - libDriver-component-c++.so
5. Create driver.cca (Ccaffeine-specific)

Driver SIDL Definition

- Driver implements standard gov.cca.ports.GoPort
- No additional interfaces defined

```
version tutorial 1.0;

package tutorial {

  class Driver implements-all gov.cca.ports.GoPort,
                              gov.cca.Component

  {
  }

}
```

Driver Component: Framework Interaction

```

tutorial::Driver_impl::setServices (
/*in*/ gov::cca::Services services )
throw ()
{
// DO-NOT-DELETE splicer.begin(tutorial.Driver.setServices)
frameworkServices = services; ← Save a pointer to the Services object
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 name
← Port type
← TypeMap pointer

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

Driver Component: GoPort implementation

```

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

// Ports
::gov::cca::Port port;
::integrators::Integrator integrator;
← Get an Integrator reference

port = frameworkServices.getPort("IntegratorPort");
integrator = port;
← Invoke the integrate method

value = integrator.integrate (lowerBound, upperBound, count);
← Output integration result

fprintf(stdout,"Value = %f\n", value);
frameworkServices.releasePort ("IntegratorPort");
return 0;
← Release ports

// DO-NOT-DELETE splicer.end(tutorial.Driver.go)
}
}
    
```

Build Issues

- Dynamic (shared) libraries
 - For each component or a set of components, build a dynamic library
 - No linking of libraries for components on which current component depends
 - Non-component libraries on which a component depends directly (e.g., BLAS), must be linked explicitly when the shared library is created

Complete Makefile for MonteCarloIntegrator

```
include babel.make

WRAPPERS = integrators_Integrator_wrapper_Impl.cc

INCLUDES = -I$(BABEL_ROOT)/include -I-$(MPI_HOME)/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)

OBJS = $(IMPLSRCS:.cc=.o) $(IORSRCS:.c=.o) $(SKELSRCS:.cc=.o) \
        $(STUBSRCS:.cc=.o) $(WRAPPERS:.cc=.o)

LIBS = -Wl,-rpath,$(BABEL_ROOT)/lib -L$(BABEL_ROOT)/lib -lsidl

libIntegrator-component-c++.so: $(OBJS)
    g++ -shared $(INCLUDES) $(OBJS) -o $@ $(LIBS)

clean:
    $(RM) *.o libIntegrator-component-c++.so
```

Running the Example

Next: Using the Ccaffeine framework



Introduction to the Ccaffeine Framework

CCA Forum Tutorial Working Group

[http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org](http://www.cca-forum.org/tutorials/tutorial-wg@cca-forum.org)

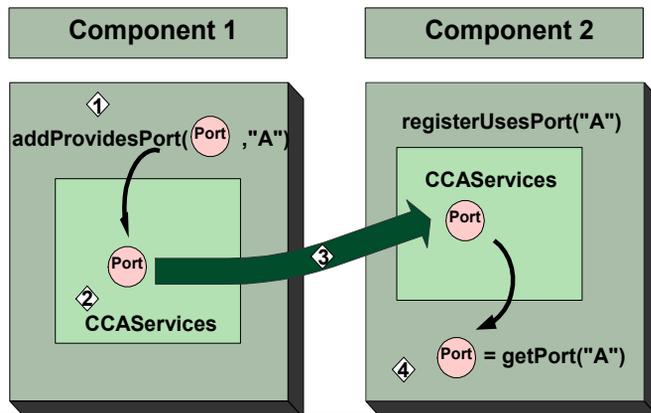


Outline

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

CCA What CCA compliant framework is expected to do ...

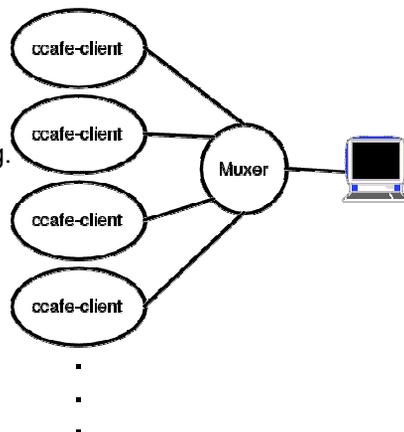
- Exchange interfaces among components without one component needing to know more about the other than the interface itself.



3

Interactive Parallel Components: what Ccaffeine does

- Executable `ccaffe-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/>

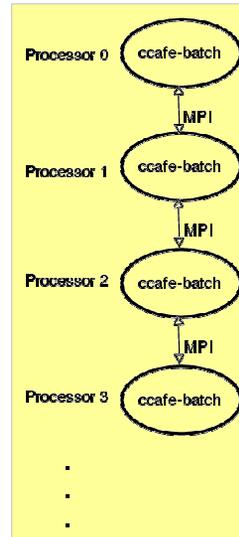
4

Ccaffeine comes in two other flavors* and a GUI.

- Single process executable: `ccafe-single`
 - really useful for debugging



- Batch executable: `ccafe-batch`
 - when all you want to do is run it.



*flavor: same executable, different name and behavior.

How to build Ccaffeine

- Have a look at <http://www.cca-forum.org/ccafe>
 - Obtain the required packages
 - Ccaffeine tar ball download
 - gcc (2.95.3, 2.96, *not* 3.x)
 - Java (>jdk1.2)
 - BLAS, LAPACK (any recent)
 - BOOST headers
 - Babel
 - Ruby (any recent, if you have Linux, probably there now)

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”

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

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

```
(cd ./dccafe; ./configure --with-cca-babel=`pwd`../cca-spec-babel \  
--with-cca-classic=`pwd`../cca-spec-classic \  
--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)
```

7

Ccaffeine build (cont'd)

- The Ccaffeine make will take ~5-10 min.
- Look in:
<http://www.cca-forum.org/ccafe/build-log.html>
for a complete listing from Rob's laptop.

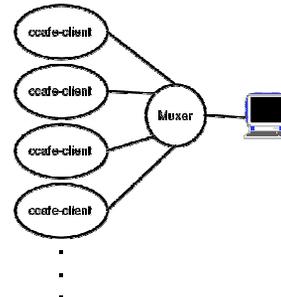
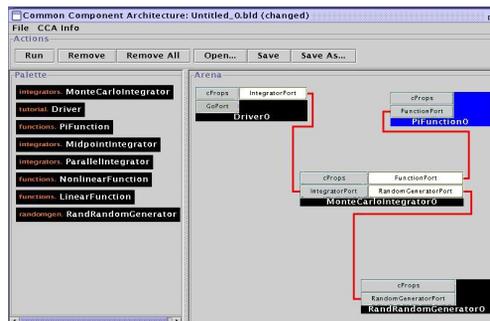
If successful you should get:

```
=====
Testing the Ccaffeine build ...
didn't crash or hang up early ... looks like it is working.
done with Ccaffeine tests.
=====
```

8

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



Ccaffeine scripting language is for those who have grown tired of the GUI

- look in:
 - http://www.cca-forum.org/ccafe/ccafe-man/Ccafe_Manual.html for all the commands
- The GUI is just a pretty front end that speaks this scripting language to the backend

You can talk directly to Ccaffeine by **typing**:

```
prompt> ccafe-single
```

```
MPI_Init called in CmdLineClientMain.cxx
```

```
my rank: 0, my pid: 25989
```

```
... (output cruft deleted)
```

```
cca>help
```

```
(complete listing of commands and what they do)
```

Quick run-through of the Ccaffeine scripting language

- Scripting language does everything that the GUI does
- **Warning:** there are two of files that Ccaffeine uses:
 - “rc” and script files for building and running apps
 - GUI “.bld” files that are state saved by the Ccaffeine GUI.

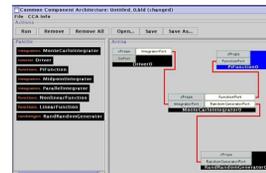
These are not the same and will give, sometimes spectacular, undefined behavior.

Magic number and repository function: the top of the script

- Must tell the framework where the components are (“path”) and which ones you want loaded into the “palette”

```

#!ccaffeine bootstrap file.
# ----- don't change anything ABOVE this line.-----
# where to find components:
path set /home/rob/cca/component
# load components into the “palette”
repository get functions.PiFunction
repository get integrators.MonteCarloIntegrator
repository get integrators.MidPointIntegrator
repository get integrators.ParallelIntegrator
repository get randomgen.RandRandomGenerator
repository get tutorial.driver
  
```

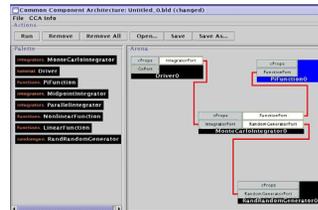


- At this point no components are instantiated, but are simply known to the system

Now start instantiating the components that will form your application

- Use the “create” function to make an instance of a component and name it
 - first arg is the class name of the component and the second is the instance name you want it to have:

```
# Instantiate and name components that have been made
# known to the framework
create randomgen.RandRandomGenerator rand
# f(x) = 4.0/(1 + x^2)
create functions.PiFunction function
create tutorial.Driver driver
```



13

Connect the components to form a complete application

- Connect takes 4 arguments, all of them are instance names of components or ports. In order they are:
 1. Using component instance name (named in “create”)
 2. Uses port instance name (name given to it by the component)
 3. Providing component instance name
 4. Provides port instance name
- Script from our example code:

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

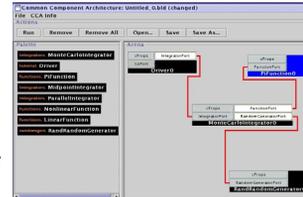
14

Time to see if it works: the “go” command

- The “go” command takes a component instance and a port instance name as an argument
 - only the named port on the named component are `go()`’ed:

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

- 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



15

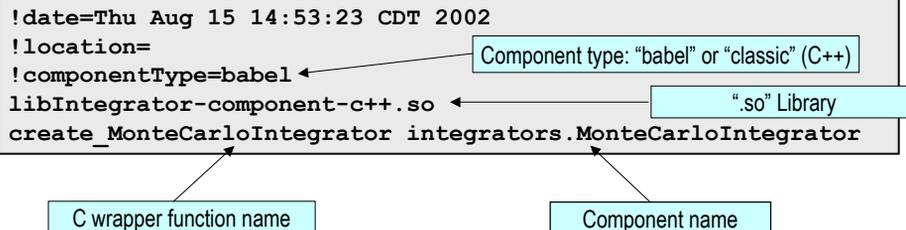
CCA is working on a component delivery specification, until then Ccaffeine has some specific req'ts

- “.cca” file describes what the format of the component is: “Babel”, or old-style “Classic.”
- Component wrapper class
 - introduces to the framework one or more components
 - contained in the “.so” file with the component(s)
 - *will go away for Babel components*

16

Example “.cca” file: MonteCarloIntegrator in integrators.cca

- Ccaffeine-specific file specifying the name of the dynamic library and creation method for each component



Wrapper C functions

- Auto-gen the wrapper C code file:
 - “genDL” scripts provided by Ccaffeine.
 - genDLWrapperStrict to generate the “.cca” file.
 - usage: genDLWrapper <component class name>
- Creates the appropriate symbols to be included in the “.so” file so that Ccaffeine can find and instantiate the component
- *In the case of Babel components this step is unnecessary and is soon to be removed*

What you are able to do now that you couldn't before ...

- Run on parallel cluster or proprietary machine with CCA components that you didn't write
 - Steve Jobs: “the best software is software I didn't have to write” – not that he actually ever did
- Develop incrementally & interactively in serial and *parallel*
 - Detach, go have lunch and reattach

Showing How it All Works

The Scripts

Next: Complex CCA Applications

setup_path script

Included in other scripts to setup paths to components and libraries.

```
export SIDL_DLL_PATH=""

DL_PATH=$BABEL_ROOT/lib
for i in `ls $TUT_COMP_ROOT | grep component`; do
  DL_PATH=$DL_PATH:$TUT_COMP_ROOT/$i ;
done

export LD_LIBRARY_PATH=$DL_PATH
export SIDL_DLL_PATH=`echo $LD_LIBRARY_PATH | sed 's/;/g'`
```

1

run_cmdline script

- Run a single interactive job using command line.
- Format: run_cmdline <full_path_to_rc_file>

dot

```
#!/bin/sh

export TUT_COMP_ROOT=`pwd`
• $TUT_COMP_ROOT/setup_path

$CCAFE_HOME/cxx/dc/user_iface/ccafe-single --ccafe-rc $1
```

2

run_gui script

- Run a single interactive job using the GUI
- Format: `run_gui <full_path_to_gui_rc_file>`

```
#!/bin/sh
export TUT_COMP_ROOT=`pwd`

• $TUT_COMP_ROOT/setup_path

$CCAFE_HOME/cxx/dc/user_iface/ccafe-single --type server --port 3314 \
--ccafe-rc $1 &
sleep 2
$CCAFE_HOME/bin/runGUI --builderPort 3314
```

3

run_gui_parallel script (1)

```
#!/bin/sh -f
# Format: run_gui_parallel <num_proc> <full_path_to_gui_rc_file>
# Configuration stuff
export mpirun=/usr/local/bin/mpirun
export CLASSPATH=$CCAFE_HOME/java:$CLASSIC_CCA_ROOT/java
export gui=$CCAFE_HOME/bin/runGUI
export TUT_COMP_ROOT=`pwd`

export javaopts=" -Djava.compiler=NONE -classpath $CLASSPATH"
export CCAFFEINE_OUT_DIR=`pwd`
export procfile="/tmp/processors.$$"
export machfile="/tmp/machines.$$"

# This tells CCAFFEINE to put the frameworks output streams
# into the current directory, into files named pOutN and
# pErrN, where N is the process number (starting from 0).
echo "Look for application output in pOut[01] and pErr[01] in
this directory"
```

4

run_gui_parallel script (2)

```
# Create a 'processors' file to tell the framework where to
# find itself and the GUI.
echo 127.0.0.1 server > $procfile
i=0
while [ $i -lt $1 ]
do
    echo $i      client >> $procfile
    i=`expr $i + 1`
done
# Create the mpirun machines file
echo 127.0.0.1 > $machfile
# Start the GUI and wait briefly to give it a chance to
# initialize
echo Launching multiplexer...
java $javaopts \
    gov.sandia.ccaffeine.dc.distributed.MuxingProcess \
    --name 127.0.0.1 --timeout 0 --file $procfile &
sleep 3
```

5

run_gui_parallel script (3)

```
# Launch the framework
echo Launching framework...
$mpirun -np $1 -machinefile $machfile $TUT_COMP_ROOT/run_client\
    --file $procfile --ccafe-rc $2 &
sleep 5
# Launch GUI
echo Launching GUI...
$gui

# Look for any stray files or processes
echo Cleaning up
rm -f $procfile $machfile
# This may be overkill
killall ccafe-client\
    gov.sandia.ccaffeine.dc.distributed.MuxingProcess \
    runGUI \
    gov.sandia.ccaffeine.dc.user_iface.BuilderClient java
```

6

run_client script

```
#!/bin/sh
#### TUT_COMP_ROOT cannot be `pwd` for mpirun reasons #####
export TUT_COMP_ROOT=/home/elwasif/CCA/tutorial/src/sidl
• $TUT_COMP_ROOT/setup_path

$CCAFE_HOME/cxx/dc/user_iface/ccafe-client $*
```

tutorial_rc_gui Ccaffeine rc file

```
#!/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
repository get integrators.MonteCarloIntegrator
repository get integrators.MidpointIntegrator
repository get integrators.ParallelIntegrator
repository get tutorial.Driver
```



A Look at More Complex Component-Based Applications

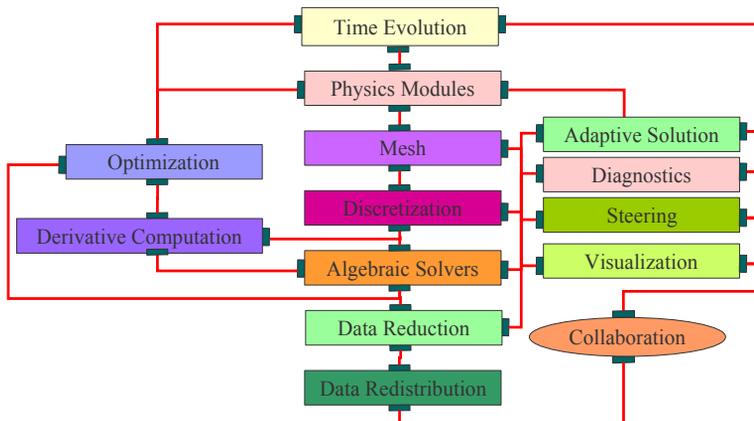
CCA Forum Tutorial Working Group

<http://www.cca-forum.org/tutorials/>
tutorial-wg@cca-forum.org



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



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

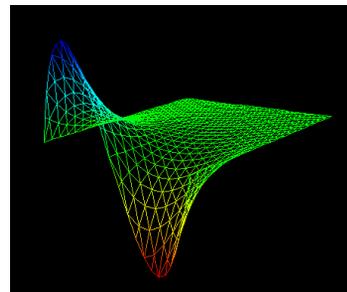
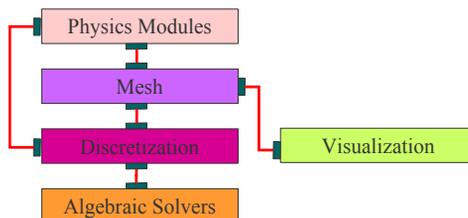
3

Our Starting Point

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

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

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



4

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

5

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

6

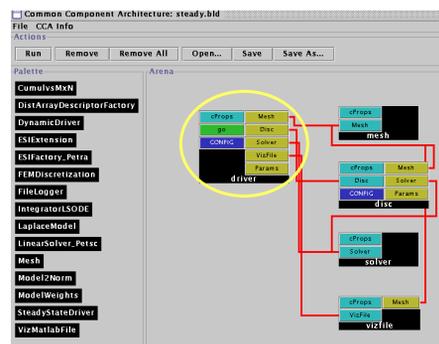
Creating the Components: Step 2

- 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 ;-)

7

The Componentized Example

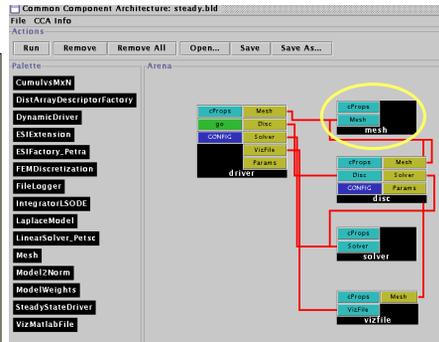
- The Driver Component
 - Responsible for the overall application flow
 - Initializes the mesh, discretization, solver and visualization components
 - Sets the physics parameters and boundary condition information



8

The Componentized Example

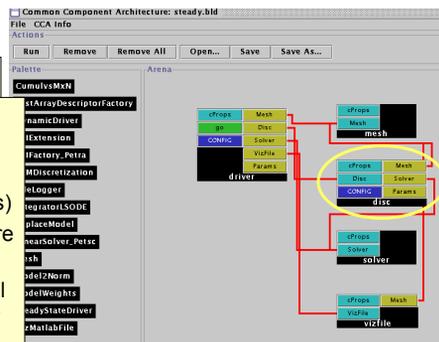
- The Driver Component
 - The Mesh Component
 - Provides geometry, topology, and boundary information
 - Provides the ability to attach user defined data as tags to mesh entities
 - Is used by the driver, discretization and visualization components



9

The Componentized Example

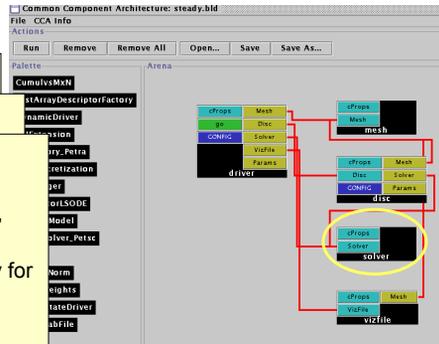
- The Driver Component
 - The Mesh Component
 - The Discretization Component
 - Provides a finite element discretization of basic operators (gradient, Laplacian, scalar terms)
 - Driver determines which terms are included and their coefficients
 - Provides mechanisms for general Dirichlet and Neumann 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)



10

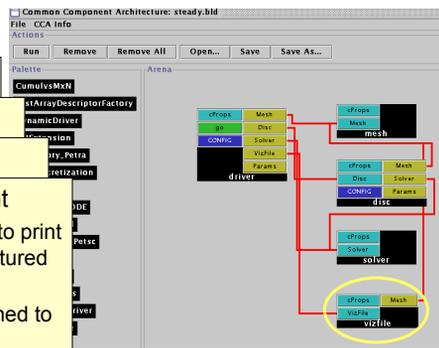
The Componentized Example

- 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



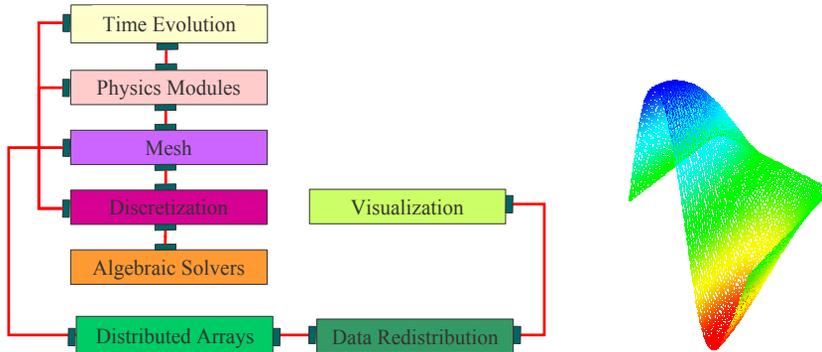
The next step... time dependence

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

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

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

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



13

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

14

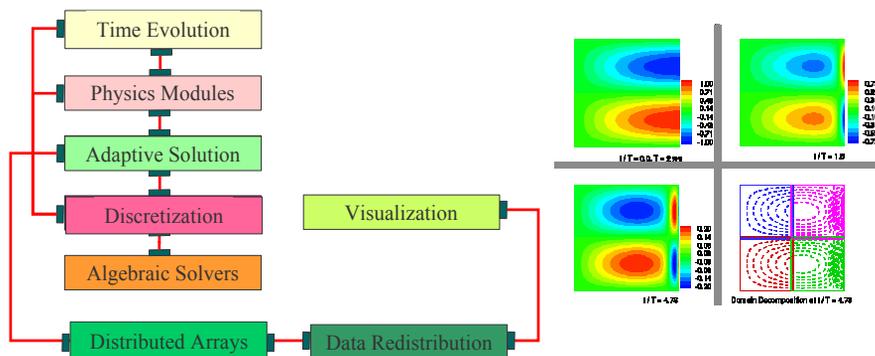
What did this exercise teach us?

- 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

17

One more layer of complexity... AMR

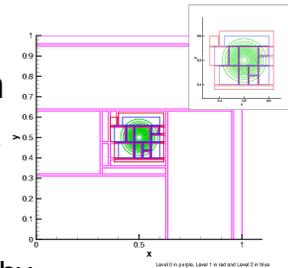
The same physics but use a block structured adaptive mesh



18

Adaptive Mesh Refinement

- Used to accurately capture a wide spectrum of length scales
- Many different techniques
 - We use structured axis-aligned patches
 - Provided by the GrACE library
- Start with a uniform coarse mesh
 - Identify regions needing refinement
 - Collate into rectangular patches
 - Impose finer mesh in patches
 - Recurse and obtain a mesh hierarchy



19

Some things change...

- The mesh component changes
 - Block structured AMR based on GRACE
- The discretization component changes
 - Finite difference on patches
 - BC handled differently
- The driver component changes

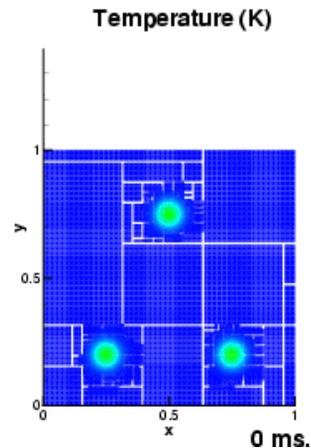
20

Beyond the heat equation...

- Flame Approximation
 - H₂-Air mixture; ignition via 3 hot-spots
 - 9-species, 19 reactions, stiff chemistry
- Governing equation

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

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



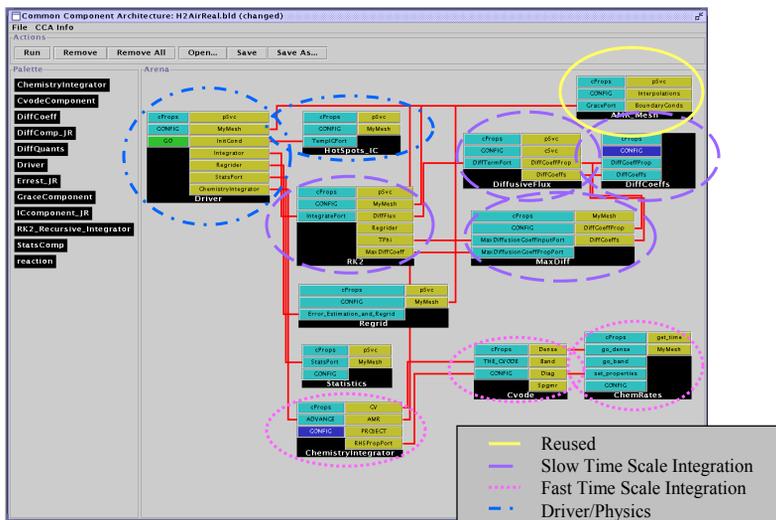
23

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%

24

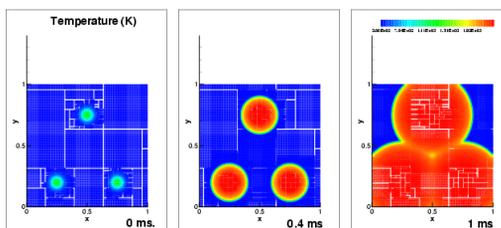
The CCA Wiring Diagram



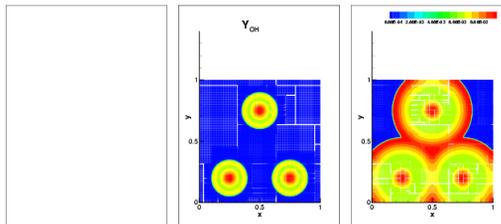
25

Evolution of the Solution

Temperature



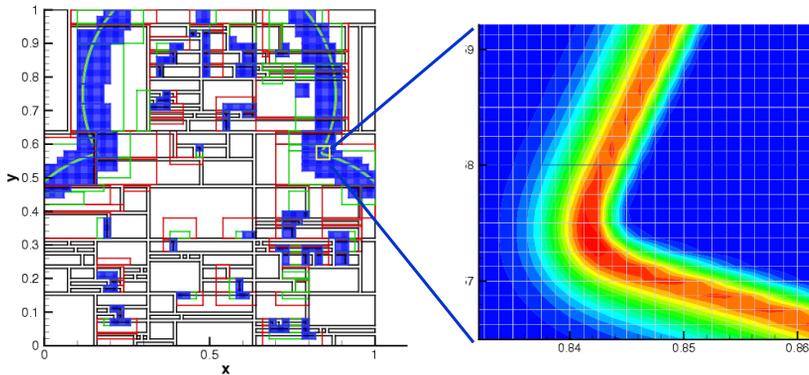
OH Profile



26

The need for AMR

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

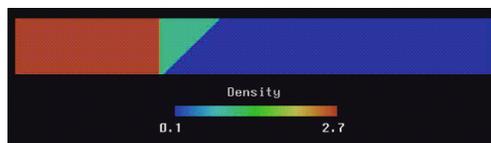


27

Shock-Hydrodynamics

- Governing equation

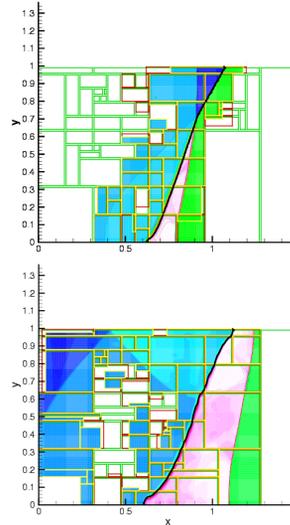
$$U_t = F_x(U) + G_y(U) \quad U = \{\rho, \rho u, \rho v, \rho E, \rho \zeta\}$$
- Domain
 - Square cross section shock-tube
- Experiment
 - Two gases are separated by a clean interface
 - Shock moves from left to right and interacts with the interface
 - Deposits vorticity
 - Reflects
 - Refracts



28

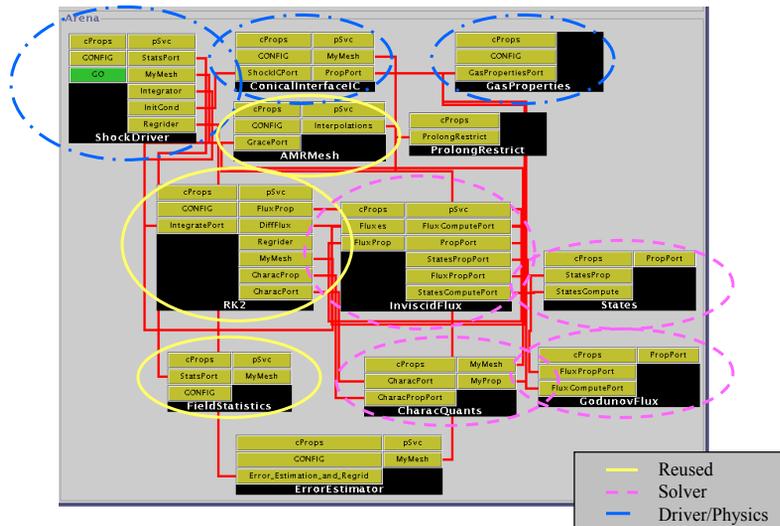
Interesting features

- Shock & interface are sharp discontinuities which need refinement
- Shock deposits vorticity – a governing quantity for turbulence, mixing, ...
- If there is insufficient refinement
 - under predict vorticity
 - slower mixing/turbulence.



29

The CCA Wiring Diagram

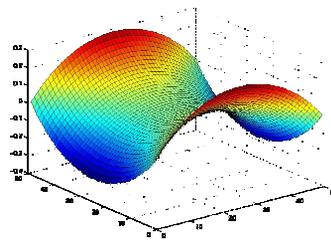


30

Unconstrained Minimization Problem

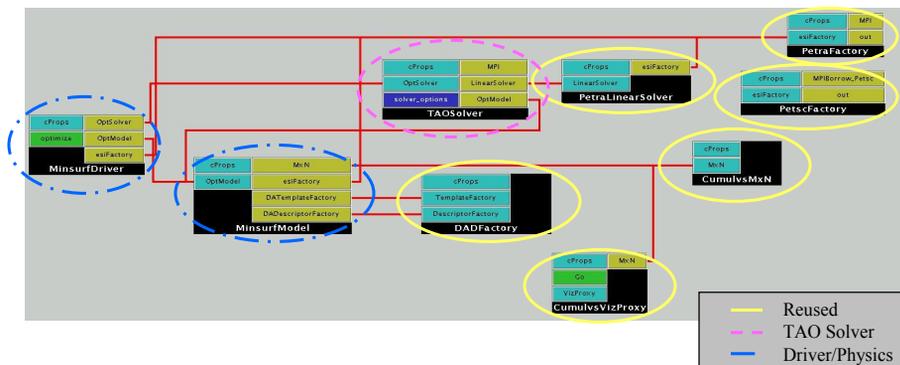
- Given a rectangular 2-dimensional domain and boundary values along the edges of the domain
- Find the surface with minimal area that satisfies the boundary conditions, i.e., compute

$$\min f(x), \text{ where } f: \mathbb{R}^n \rightarrow \mathbb{R}$$
- Solve using optimization components based on TAO (ANL)



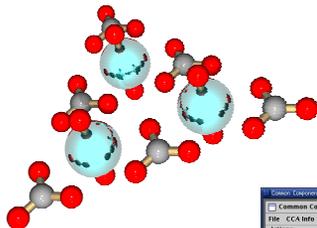
31

Unconstrained Minimization Using a Structured Mesh



32

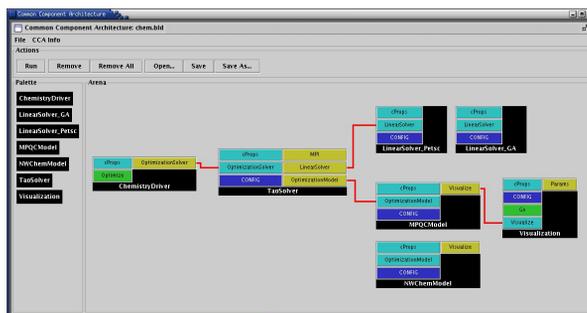
Molecular Geometry Optimization



Relativistic quantum chemistry calculation of $(\text{UO}_2)_3(\text{CO}_3)_6$ using NWChem. Image courtesy of Wibe deJong, PNNL.

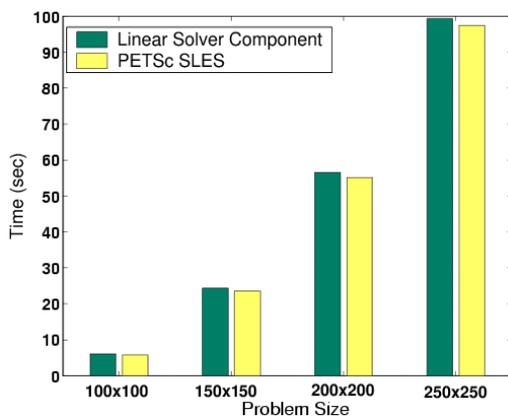
Wiring diagram using Ccaffeine framework and:

- Electronic structure components based on NWChem (PNNL) and MPQC (SNL)
- Optimization components based on TAO (ANL)
- Linear algebra components based on Global Arrays (PNNL) and PETSc (ANL)



33

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

34

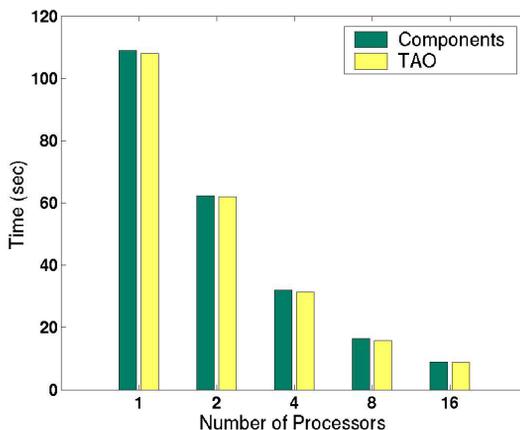
Overhead from Component Invocation

- 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

35

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

36

List of Component Re-Use

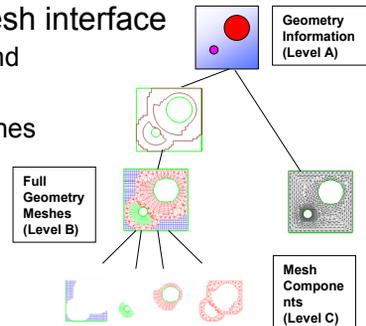
- Various services in Ccaffeine
 - Integrator
 - *IntegratorLSODE* (2)
 - *RK2* (2)
 - Linear solvers
 - *LinearSolver_Petra* (4)
 - *LinearSolver_PETSc* (4)
 - AMR
 - *AMRmesh* (3)
 - Data description
 - *DADFactory* (3)
 - Data redistribution
 - *CumulvsMxN* (3)
 - Visualization
 - *CumulvsVizProxy* (3)
- } Component interfaces to numerical libraries
- } Component interfaces to parallel data management and visualization tools

37

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

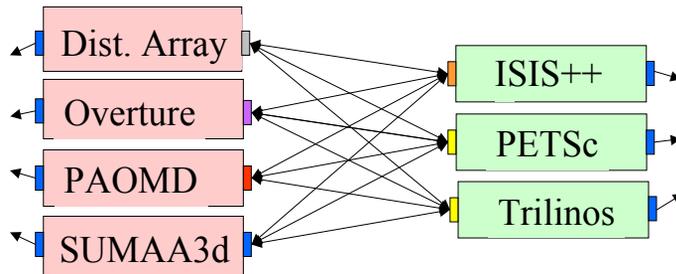


38

Current Situation

Current Situation

- Public interfaces for numerical libraries are unique
- *Many-to-Many* couplings require *Many²* interfaces
 - Often a heroic effort to understand the inner workings of both codes
 - Not a scalable solution

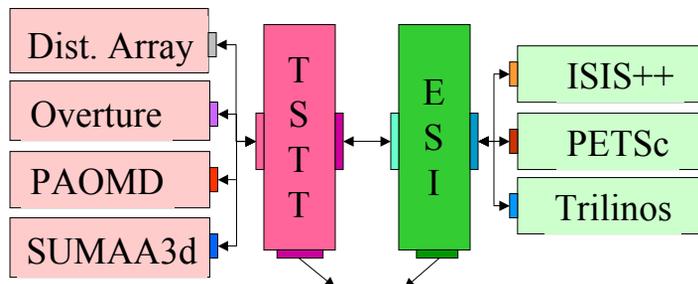


39

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



40

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

41

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

42

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

43

Summary

- Complex applications that use components are possible
 - Shock hydrodynamics
 - Chemistry applications
 - Optimization problems
- 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

44

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!

45

Next: **Status and Plans**

46



CCA Status and Plans

CCA Forum Tutorial Working Group

[http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org](http://www.cca-forum.org/tutorials/tutorial-wg@cca-forum.org)

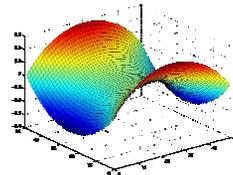
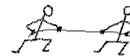


CCTTSS Research Thrust Areas and Main Working Groups

- 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
Scott Kohn, LLNL (skohn@llnl.gov)
- User Outreach
David Bernholdt, ORNL (bernholdtde@ornl.gov)

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



3

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

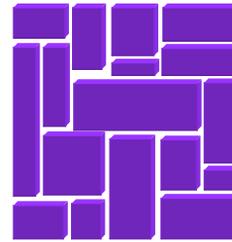
4



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, ANL
 - Collaboration with SciDAC TSTT Center
- Structured Block AMR
 - Phil Colella, LBNL
 - Collaboration with APDEC & TSTT



5

Scientific Data Interfaces

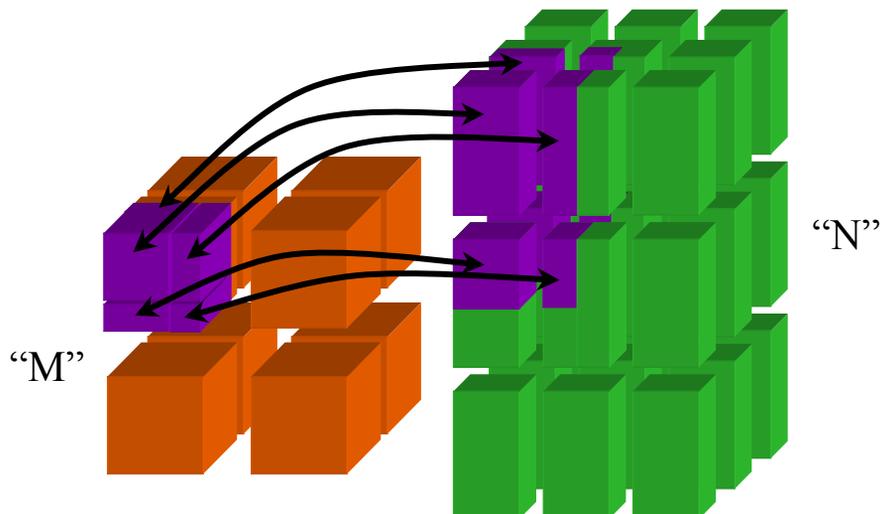
- Low Level, Raw Data
 - Supports high performance access to memory
 - Based on IOVec
 - Assumes a contiguous memory block
 - Supports basic data types such as integer, float, double
 - No topology information
- Local & Distributed Arrays
 - Abstract interfaces for higher-level data description
 - 1D,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

6

Mesh Interfaces

- Unstructured Meshes
 - Abstract interfaces for mesh and geometry access and modification
 - Supports geometry and topology access via iterators, arrays, worksets
 - Separates structured and unstructured mesh access for performance
- Block Structured AMR
 - Abstract interfaces for allowing block structured AMR packages to exchange data

“MxN” Parallel Data Redistribution: The Problem...



“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”
 - $M \neq N$ ~ Distinct, Pronounced “M by N”...
 - Model coupling, e.g., climate, solver / optimizer
 - Collecting data for visualization (“Mx1”)
- Define “standard” interface
 - Fundamental operations for any parallel data coupler
 - Full range of synchronization and communication options

9

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
 - Units, time & grid Interpolation, flux conservation
- “Automatic” MxN Service via Framework
 - Implicit in method invocations, “parallel RMI”



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

10

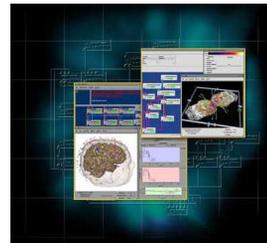
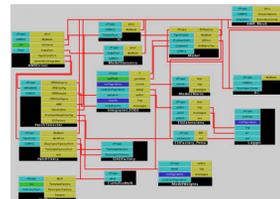
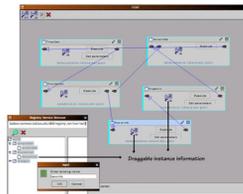
CCA Frameworks

- Component Containers & Run-Time Environments
- Research Areas:
 - Integration of prototype frameworks
 - SCMD/parallel with distributed
 - 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

11

CCA Framework Prototypes

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



12

Outreach and Applications Integration



- 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!

Current CCA / CCTTSS Status

- CCA Specification at Version 0.5
- Several Working Prototype Frameworks
- Functional Multi-Component Parallel and Distributed Demonstration Applications
- Draft specifications for
 - Basic scientific data objects
 - MxN parallel data redistribution
- Demonstration Software **Available for Download**
 - 4 different “direct connect” applications, 1 distributed
 - 31 distinct components, up to 17 in any single application, 6 used in more than one application

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/>
 - <http://www.cca-forum.org/software.html>
 - tutorial-wg@cca-forum.org
 - cca-forum@cca-forum.org

