The Common Component Architecture

A Component Environment for High Performance Scientific Computing

David E. Bernholdt
Oak Ridge National Laboratory

In collaboration with the CCA Forum

bernholdtde@ornl.gov

Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, U.S. Dept. of Energy
What is the CCA?

- A component model specifically designed for high-performance scientific computing
- Supports both parallel and distributed applications
- Designed to be implementable without sacrificing performance
- Minimalist approach makes it easier to componentize existing software

- A tool to enhance the productivity of scientific programmers
  - Make the hard things easier, make some intractable things tractable
  - Not a magic bullet
Origins of the CCA

- DOE Advanced Computational Software (ACTS) program
  - improve interoperability and reuse of major DOE software libraries/tools
- Pair-wise ($N^2$) approach to interoperability
  - Not scalable
- CCA Forum
  - Grassroots effort to improve on ACTS via component approach
  - Launched in 1998
  - Standalone funding through DOE Scientific Discovery through Advanced Computing (SciDAC) program in 2001
SciDAC ISICs: An Aside

- **Scientific Discovery through Advanced Computing**
- **Integrated Software Infrastructure Centers**

**Creating a Scientific Computing Software Infrastructure:**
- Create a new generation of *Scientific Simulation Codes* that take full advantage of the extraordinary computing capabilities of terascale computers.
- **Create the Mathematical and Computing Systems Software** to enable Scientific Simulation Codes to effectively and efficiently use terascale computers.
- Create a *Collaboratory Software Environment* to enable geographically-separated scientists to effectively work together as a team to facilitate remote access to both facilities and data.
The SciDAC ISICs

- The Common Component Architecture (CCA)
- High-End Computer System Performance: Science and Engineering (PERC)
- Scalable Systems Software Center (SSS)
- Scientific Data Management Center (SDM)
- Algorithmic and Software Framework for Applied Partial Differential Equations (APDEC)
- Terascale Optimal PDE Simulations Center (TOPS)
- Terascale Simulation Tools and Technology Center (TSTT)
- Programming Models (PModels)
Some Productivity Issues in Scientific Computing

- **Reuse & Interoperability**
  - Few domains share/reuse code significantly
  - Standardization of interfaces rare outside of linear algebra tools

- **Multiple languages**
  - Fortran still widely used
  - C++, Java, scripting languages increasingly popular

- **Size/complexity of code**
  - Including support of diverse parallel architectures

- **Coupling of codes (multi-scale, multi-physics, etc.)**
  - Increasingly interesting as computer power increases
  - Current practice: loose coupling (i.e. files), monolithic codes, no generality of interfaces between codes
Addressing Productivity Issues with Components

• Reuse & Interoperability
  – Component models *support and promote* R & I
  – Interface standardization still very important, burden on user

• Multiple languages
  – CCA developing Babel: Fortran, C, C++, Java, Python as peers

• Size/complexity of code
  – Component models promote a Lego block/”plug and play” approach to large codes

• Coupling of codes (multi-scale, multi-physics, etc.)
  – Natural extension of component-based applications
CCA and Commodity Component Environments

• “Commodity” environments include CORBA, COM, Enterprise JavaBeans

• Minimize adoption overhead
  – Make it easy to componentize legacy software

• Minimize performance impact
  – Allow tightly-coupled in-process components

• Support for tightly-coupled parallel computing
  – Commodity envs are focused on distributed
  – Distributed is nice, but parallel is critical

• Support languages and types (and platforms) for science
  – Fortran, complex numbers, arrays, etc.
Basic CCA Terminology

• **Port** (aka *interface*)
  – Procedural interface (not just dataflow!)
  – Like C++ abst. virtual class, Java interface
  – Uses/provides design pattern

• **Component**
  – A unit of software deployment/reuse (i.e. has interesting functionality)
  – Interacts with the outside world only through well-defined interfaces
  – Implementation is opaque to the outside world

• **Framework**
  – Holds components during application composition and execution
  – Controls the “exchange” of interfaces between components (while ensuring implementations remain hidden)
  – Provides a small set of standard services to components
    - *CCA spec doesn’t specify a framework per se, so components can be constructed to provide framework-like services*
Existing Code → Components

- 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

Framework interaction code *(new)*

Integrator library code *(slightly modified)*

Integrator
Framework Mediates Component Interactions

CCA.Services provides Result
uses Fun

CCA.Services provides Fun

Function code
Fun(x) = 3 * x + 17

Integrator

getPort(Fun)
y=Fun(x)
releasePort(Fun)

LinearFunction
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
“Direct Connection” Maintains Local Performance

• 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 are in other direct connect fwks
Framework Stays “Out of the Way” of Component Parallelism

• Single component multiple data (SCMD) model is component analog of widely used SPMD model

• Each process loaded with the same set of components wired the same way

• Different components in same process “talk to each” other via ports and the framework

• Same component in different processes talk to each other through their favorite communications layer (i.e. MPI, PVM, GA)

Components: Blue, Green, Red
Framework: Gray

MCMD/MPMD also supported
Language Interoperability

- Existing language interoperability approaches are “point-to-point” solutions
- Babel provides a unified approach in which all languages are considered peers
- Babel used primarily at interfaces
Babel Features

- **Babel includes...**
  - Code generator
  - Runtime (linked into CCA framework)
- Implemented using C-based internal object representation (IOR)
- **Server side:** wrap implementation in Babel-generated code to allow calling from any language via IOR
- **Client side:** Use SIDL interface definition to generate stubs to call from client language to IOR
- Strives to allow natural-looking code in each supported language

**Scientific Interface Def. Lang. (SIDL)**
- **Objects:** Interfaces, Abstract Classes, Concrete Classes
- **Methods:** all public; virtual, static, final
- **Mode:** in, out, inout (like CORBA)
- **Types:** bool, char, int, long, float, double, fcomplex, dcomplex, array<Type,Dimension>, enum, interface, class

**Diagram:**
- SIDL interface description ➔ Babel Compiler ➔ C Stubs, IORs, C Skels, C Impls ➔ libfunction.so
CCA Research Thrusts and Application Domains

- **Frameworks**
  - Framework interoperability
  - Language interoperability
  - Deployment
- **Scientific Components**
  - Data Components
  - Linear Algebra
  - Visualization & Steering
  - ...
- **MxN Parallel Data Redistribution**
  - Component-based
  - Framework-based
- **Application Outreach**
  - Education
  - Best practices for use
  - Chemistry, Climate
- **Chemistry**
- **Combustion**
- **Climate Modeling**
- **Meshing Tools**
- **(PDE) Solvers**
- Supernova simulation
- Accelerator simulation
- Fusion
- ASCI C-SAFE
- …
Measuring the Success of the CCA

- Increasing programmer productivity
  - More reuse
  - Development and use of domain-specific interfaces
  - Cross-pollination between fields (i.e. combustion components used in supernova simulation app)

- Serious high-performance component-based applications
  - i.e. CCA-based apps contending for Gordon Bell Awards, achieving high sustained performance, etc.
  - Shows that CCA doesn’t “get in the way”

- Researchers undertaking projects on a scale they would not have previously considered
  - And retaining their sanity
Summary and Contacts

• CCA is a component environment for high performance scientific computing
  – Parallelism
  – Performance
  – Language support
  – Ease of adoption

• Increasing programmer productivity is fundamental goal

• Web site: http://www.cca-forum.org

• Rob Armstrong <rob@sandia.gov>
  – CCA Forum head and SciDAC Center Lead PI

• David Bernholdt <bernholdtde@ornl.gov>
  – co-PI, Lead for User Outreach and Applications Integration