An Overview of the Common Component Architecture (CCA)

See companion presentation:

*How the Common Component Architecture Advances Computational Science*
Motivation

- Complexity of scientific software increases with simulation fidelity, multi-physics coupling, computer power → *software crisis*

- Component technology is well established outside high-performance computing (HPC) as a way to manage software complexity
  - All enterprise software is component software, but commercial implementations do not support HPC

- **Common Component Architecture (CCA)** brings a component software approach to scientific HPC
  - Grassroots effort launched in 1998
Benefits to software developers

- Components are natural units of decomposition and interaction for both software and developers
  - Manage software complexity
- They enable scientists to work together as a cohesive scientific enterprise, across disciplines, geographical boundaries, and technical preferences by facilitating...
  - collaboration around software development
  - interoperability and reuse of software tools
  - community standards for scientific software
  - coupling of disparate codes

CCA-based simulation of OH concentration in advective-diffusive-reactive simulation using 4th order Runge-Kutta-Chebyshev integrator on four levels of adaptively refined mesh

Courtesy of J. Ray, Sandia National Laboratories
Basic CCA concepts

- **Components**
  - Are units of software development/functionality
  - Interact only through well-defined interfaces
  - Can be composed into applications based on their interfaces

- **Frameworks**
  - Hold components while applications are assembled and executed
  - Control the connections of ports
  - Provide standard services to components

- **Ports**
  - Are the interfaces through which components interact
  - Follow a provides/uses pattern
    - Provided ports are implemented by a component
    - Used ports are functionality a component needs to call

---

Screenshot of application in the Ccaffeine framework’s GUI
CCA features for scientific HPC

• Parallel computing
  – Component mechanisms apply within a process
  – Parallelism across processes is up to each component
    • Usual tools: MPI, Global Arrays, PVM, ...
  – Both SPMD and MPMD supported

• Distributed computing
  – Supported transparently to components

• Performance
  – Components in same process share memory
    • Small overhead on inter-component calls
  – No overhead on parallel communication
  – Minimal language interoperability overhead

• Language interoperability
  – Implementation language of component should not matter to others
  – Babel treats all supported languages as peers
  – SIDL allows language-neutral specification of interfaces
User-friendly applications in CCA

• Bocca build and development tools
  – Script-based environment for defining CCA applications
    • Accommodates legacy/existing as well as new applications
  – Automates many tedious and error-prone tasks
    • Automatically generates CCA boilerplate code

• Cut through complexity via higher-level functions
  – Manage SIDL-based projects containing enums, interfaces, classes, ports, and components
  – Reliably edit the user implementation/code sections
  – Automated GNU Make–based build system
  – Examples:
    `bocca create project MyProj ; cd MyProj ; ./configure`
    `bocca create port MyPort ; bocca edit MyPort ...`
    `bocca create component MyComp --uses=MyPort@out --provides:MyPort@in`

• Bocca stays out of the way of code evolution
  – But it is a useful tool for extending/maintaining CCA codes
Current status of the CCA

• CCA specification well established and stable
  – Approaching “1.0” completeness

• Suite of tools implements the CCA environment
  – Babel (language interop), Bocca (build/development)
  – Ccaffeine (framework)
    • Other frameworks also available

• CCA tools and concepts are used by 25+ different application groups in diverse fields
  – CCA provides a common infrastructure for developing simulation toolkits and frameworks, coupling disparate codes, and many other types of applications
  – CCA benefits users in many different ways
  – See companion presentation How the Common Component Architecture Advances Computational Science
CCA research and development plans

• Leverage the component environment to provide important new capabilities to software developers
  – Adapt running applications for performance, accuracy, faults, and other criteria
  – Improve software quality via software contracts, testing, and verification
  – Use high-end hardware with massive parallelism, heterogeneous processors

• Mature the CCA environment and tools to production quality

• Grow a “component ecosystem”
  – Enable plug-and-play application development using off-the-shelf scientific components

• Help computational scientists effectively use component technology
The CCA community

- The CCA Forum is the standards body and user group
  - Quarterly face-to-face meetings, mailing lists, collaboration resources

- Many other projects and sponsors contribute to development and use of CCA
For more information

- See companion presentation: *How the Common Component Architecture Advances Computational Science*

- ORNL booth at SC2010
  - David E. Bernholdt, Wael Elwasif, Samantha Foley (ORNL)

- Other booths at SC2010
  - Ames Laboratory (2705), Fang (Cherry) Liu, Masha Sosonkina, and Theresa Windus
  - Argonne National Laboratory (2513), Lois McInnes, Boyana Norris
  - NNSA/ASC (2438), Sameer Shende
  - Tech-X Corporation (2950), Steve Goldhaber, Sveta Shasharina, Nanbor Wang

- On the internet
  - http://www.cca-forum.org
  - cca-forum@cca-forum.org