Enabling Advanced Scientific Computing Software

Steven Parker, University of Utah
Rob Armstrong, Sandia National Laboratory
David Bernholdt, Oakridge National Laboratory
Tamara Dahlgren, Lawrence Livermore National Laboratory
Tom Epperly, Lawrence Livermore National Laboratory
Joseph Kenny, Sandia National Laboratory
Manoj Krishnan, Pacific Northwest National Laboratory
Gary Kumfert, Lawerence Livermore National Laboratory
Jay Larson, Argonne National Laboratory
Lois Curfman McInnes, Argonne National Laboratory
Jarek Nieplocha, Pacific Northwest National Laboratory
Jaideep Ray, Sandia National Laboratory
Sveta Shasharina, Tech-X Corporation

CTWatch QUARTERLY
November 2007

Overview

The SciDAC Center for Technology for Advanced Scientific Computing Software (TASCS) focuses on developing tools, components and best practices for developing high quality, reusable high-performance computing software. TASCS fosters the Common Component Architecture (CCA) through a community forum that involves a wide range of participants. The CCA environment aims to bring component-based software development techniques and tools, which are commonplace in the computing industry, to high performance computing. To do so, several challenges are being addressed including parallelism, performance, and efficient handling of large datasets. The CCA has produced a specification that allows components to be deployed and reused in a highly extensible yet efficient parallel environment. The primary advantage of this component-based approach is the separate development of simulation algorithms, models, and infrastructure. This allows the pieces of a complex simulation to evolve independently, thereby helping a system grow intelligently as technologies mature. The CCA tools have been used to improve productivity and increase capabilities for HPC software in meshing, solvers, and computational chemistry, among other applications.

TASCS supports a range of core technologies for using components in high-performance simulation software, including the Caffeine framework, the Babel interoperability tool, and the Bocca development environment for HPC components. In addition, the CCA helps provide access to tools for performance analysis, for coupling parallel simulations, for mixing distributed and parallel computing, and for ensuring software quality in complex parallel simulations. These tools can help tame the complexity of utilizing parallel computation, especially for sophisticated applications that integrate multiple software packages, physical simulation regimes or solution techniques. We will discuss some of these tools and show how they have been used to solve HPC programming challenges.

Component-based Software Engineering

The component-based software engineering (CBSE)[1]
methodology has been developed to facilitate the understanding, development, and evolution of large-scale software systems. By emphasizing strong encapsulation of code with well defined interfaces between modules, a component approach provides a way of decomposing software into units that are conceptually manageable, and that interact in specific and easily understood ways.

These characteristics also facilitate the design and evolution of large, complex software systems by distinguishing between the functional specification of a component (fixed or slowly changing) and its implementation (possibly more rapidly changing, or even having multiple implementations). With thoughtful design of interfaces, component approaches can promote software reuse and interoperability. The encapsulation of components makes them useful in collaborative software development situations, where individuals or small groups take responsibility for the implementation of components conforming to interface specifications agreed to by the collaboration as a whole. These characteristics match very well with the way that the modern computational science community approaches simulation software, which makes the component approach an ideal match for high-performance scientific computing. Furthermore, simulation coupling is of rapidly growing importance in scientific computing, and maps directly to the philosophy of software components.

Although the idea of CBSE has a long history, component architectures have only recently become practical for use in high-performance scientific computing. Development of the Common Component Architecture,[2] a general component environment, began with the establishment of the CCA Forum [3] in 1998. The Cactus framework,[4]
originally developed primarily to support numerical relativity simulations, began appearing in the scientific literature in roughly 1999. Another domain-specific framework effort, the Earth System Modeling Framework (ESMF),[5] [6] began in 2001.

Scientific computing poses both technical and sociological challenges to the deployment and adoption of new technologies, such as components and frameworks. The scientific CBSE community is still in the formative stages of understanding how these concepts can be used most effectively in the context of advanced computational science applications. At the same time, the field is evolving: computing power grows, the tools and software environments evolve, and applications move to take advantage of new capabilities not just to solve larger problems faster, but also at higher levels of physical fidelity. If CBSE is to become a routine part of computational science, we need to anticipate emerging trends and how they will impact the concepts and tools of CBSE. These emerging trends in high-end scientific computing pose both challenges and opportunities for component-based software development, and provide incredible opportunities to dramatically enhance the reliability, maintainability, and scope of HPC applications.

The CCA Component Model

Formally, the Common Component Architecture is a specification of an HPC-friendly component model. This specification provides a focus for an extensive research and development effort. The research effort emphasizes understanding how best to utilize and implement component-based software engineering practices in the high-performance scientific computing arena. The development effort creates practical reference implementations and helps scientific software developers use them to create CCA-compliant components and applications.

The CCA specification is expressed as a set of abstract interfaces[7]
written in the Scientific Interface Definition Language (SIDL). SIDL is used by the Babel language interoperability tool (discussed further below), which implicitly defines bindings to the various languages that Babel supports (currently Fortran 77, Fortran 90, C, C++, Python, and Java).

The primary players in a CCA application are *Components* that encapsulate a particular piece of software, *Ports* that define the interfaces between components, and *Framework*
that glue the aforementioned components together and allow them to communicate through the ports that are defined. Figure 1 illustrates how several such components combine together to form a single application. This conceptual model should be familiar to anyone that has used component-based systems before, except that the components explicitly support parallelism and the ports facilitate fine-grained communication of large quantities of data without the copying that is inherent in many such systems.

The core of the CCA specification is the *Services*
interface. This is the primary means by which components interact with the framework, allowing the component to inform the framework of component capabilities and interfaces, and to request access to other services the framework may provide, such as information about connections between itself and other components, or the ability to instantiate and otherwise manipulate other components. The Services interface allows a component to declare two different types of ports, those that it will *provide* and those that it will *use*. These ports can also be thought of as callee and caller. These ports make it possible for a CCA framework to effectively mediate connections between components, and allows components to be assembled by another entity (a user through a GUI, a script, or even another component). The CCA component model espouses a minimalist approach, requiring only that components implement a single method/function (called setServices) that establishes contact between the component and the framework.

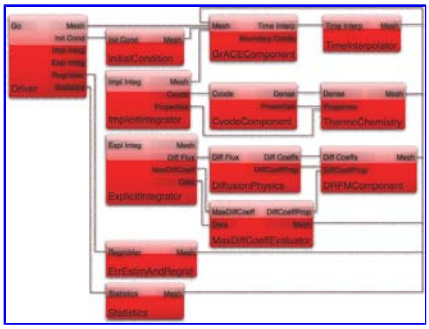

Figure 1. A CCA Component wiring diagram showing the interconnection of components in a reaction-diffusion combustion simulation.

CCA ports are simply a babel-described interface, and are also identified by a type and name. An optional set of properties associated with each port can describe additional functionality, such as the minimum/maximum number of connections. Components may provide multiple ports and even multiple instances of the same port. In addition to defining the port mechanism, the CCA specification also defines a number of specific ports that are useful in multiple applications, such as the GoPort (for starting an application), parameter ports for communicating basic configuration information to the application, and ports for communicating events to other components. The CCA reuses this port mechanism to export services provided by the framework that allow a component to assemble and manage other components, monitor available components, and watch for application events. Using this mechanism, graphical user interfaces become simply a component that is instantiated in the system and are not tied to the underlying framework. These services also allow dynamic behavior of the application itself, such as swapping components, and provide a mechanism for a hierarchy of components that are assembled at multiple levels of abstraction.

Software Tools

Beyond the core specification, a number of software tools have been developed that assist users in developing HPC applications around component technology. These tools, developed both through the TASCS Center, and through CCA collaborators, provide interoperability between programming languages, assistance with packing and deployment, and tools for performance analysis. There also exists a handful of different CCA-compliant frameworks that target different operating environments. A few of these tools are described here and additional information can be found at the CCA Forum home page.

Babel (pronounced babble)[8]
addresses the language interoperability problem using a Scientific Interface Definition Language (SIDL) that provides the ability to interact between programming languages and platforms, while addressing the unique needs of parallel scientific computing. Given a SIDL description that describes the calling interface (but not the implementation) of a particular software library, Babel generates glue code that allows software implemented in one supported language to be called from any other supported language. SIDL supports complex numbers and dynamic multi-dimensional arrays as well as parallel communication directives that are required for parallel distributed components. SIDL also provides other common features that are generally useful for software engineering, such as enumerated types, symbol versioning, and name space management, and employs an object-oriented inheritance model similar to Java. Babel provides a code splicing capability that preserves old edits during the regeneration of implementation files after modifications to the SIDL source.

Babel recently added a remote method invocation that provides a consistent mechanism to communicate between objects regardless of where they are located. This model provides a simpler and more consistent object-oriented programming model than CORBA or COM, and provides an API for third-party plug-ins to customize the underlying communication model. A simple TCP/IP protocol is provided that outperforms both CORBA and Web Services. Babel RMI fills a niche in “short-haul” distributed computing - within a machine room, or even in a single machine with concurrent MPI runs.

Ccaffeine[9]
is the main CCA framework implementation for HPC parallel computing and it supports the component analogs of both the single program/multiple data (SPMD) and multiple program/multiple data (MPMD) parallel programming models. We refer to these as single or multiple *component*/multiple data (SCMD or MCMD) models. Figure 2 depicts the SCMD case; each process is loaded with the same set of components wired together in the same way. Interactions among components within a given process (vertical direction) take place through the normal CCA means - through Ports. Interactions within a parallel component (called a parallel cohort) take place via the parallel programming model that the component uses (typically MPI). “Diagonal’” interactions - between component A on one process and component B on another process - are not prohibited by the CCA, but are currently not supported in Ccaffeine.

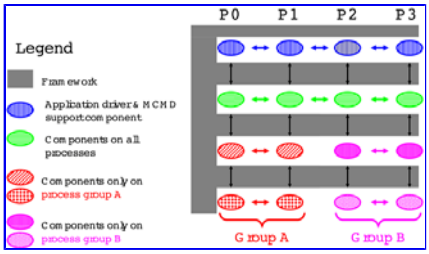

Figure 2. A schematic representation of the CCA parallel programming environment in the single component/multiple data (SCMD) paradigm.

Bocca
is a system for creating, managing, and deploying CCA-based components. Bocca can create components, define ports and interfaces, and manage the build system for the resulting component. Bocca is a new addition to the CCA tool suite, but promises to dramatically simplify the process of creating a component from scratch and subsequently maintaining it.

Performance Monitoring and Tuning. TAU is a robust and portable measurement interface and system for software performance evaluation. Using SIDL to describe TAU's measurement API, Babel has enabled access to TAU across all supported languages. CCA/Babel has also enabled incorporation of dynamic selection of measurement options into the TAU performance evaluation tools. Users can choose from a variety of measurement options interactively at runtime, without re-compilation of applications. Proxy components are automatically generated to mirror a component's interface, allowing dynamic interposition of proxies between callers and callees, via hooks into the intermediate Babel communication layer. Such inter-component interaction measurements can correlate performance to application parameters, used for constructing more sophisticated performance models.

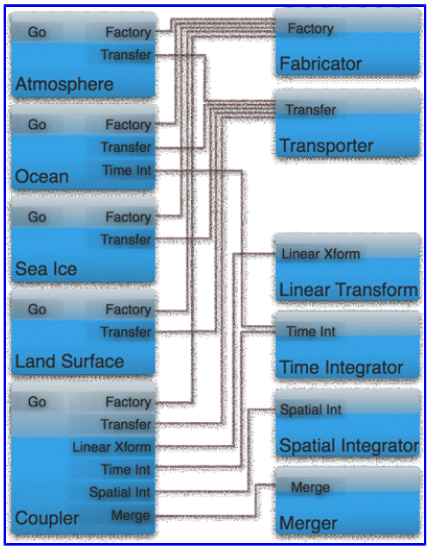

Figure 3. A Component wiring diagram showing how explicit coupling components can manage the interfaces in a multi-physics climate simulation.

Components for Parallel Coupling. Multiphysics and multiscale models face a formidable obstacle: the parallel coupling problem. Parallel coupling involves the description, transfer, and transformation of distributed data. We are developing a set of CCA components (the Parallel Coupling Infrastructure, or PCI Toolkit) that leverage successful parallel coupling technology - the Model Coupling Toolkit - to simplify the process of remapping data between disparate discretizations and processor mappings.

Additional tools. In addition to these tools and software frameworks, the TASCS Center maintains additional component-based software for developing HPC applications on CCA technology. A graphical user interface allows interactive construction and monitoring of HPC applications. CCA-lite is a slimmed down version of the CCA specification designed for statically-linked components that are written in C, C++ and/or Fortran. Additional frameworks, such as the SCIJump distributed/parallel framework from Utah and the LegionCCA Grid-based framework from SUNY Binghamton, also provide alternative deployment vehicles for CCA components.

Applications

CCA is applicable to a broad range of parallel applications. We highlight a few of these endeavors.

Combustion Modeling. One of the most sophisticated implementations of the CCA paradigm to date is in combustion modeling. The endeavor, which started in 2001 at the Computational Facility for Reacting Flow Science (CFRFS) project, [10] seeks to create a facility for the high fidelity simulation of flames involving realistic physical models, nonlinear PDEs, and a spectrum of time and length scales. Given the complexity of the problem and the multiplicity of physical and mathematical models required for the task, a component based approach was a natural fit. CCA was chosen primarily for its high performance and simplicity. General purpose components, implementing a particular numerical or physical functionality, are reused in various code assemblies.

Fusion. The standardization of fusion codes has become of paramount importance with the beginning of the fusion integrated modeling. In 2006, two SciDAC projects, SWIM (Center for Simulation of RF Wave Interactions with Magnetohydrodynamics)[11] and CPES (the Center for Plasma Edge Simulation),[12] were funded. In 2007, another integrating SciDAC project FACETS (the Framework Application for Core-Edge Transport Simulations)[13] started. Each of

these three projects addresses a different area of integration, and some of them will possibly unite in the upcoming Fusion Simulation Project sometime in 2008. All projects are looking at components technologies to assist them in defining and composing participating codes. FACETS uses the CCA language interoperability tool, Babel, to wrap F90 modules for the use in its C++ framework.

Quantum Chemistry. In response to the strong need for a community software base in the quantum chemistry community, members of the Quantum Chemistry Scientific Application Partnership (QCSAP) have leveraged the software engineering practices and tools developed by the CCA Forum to create a community-based architecture for chemistry simulation. By designing flexible interfaces by which quantum chemistry capabilities can be shared, the scaling of human effort is drastically improved, allowing the exploration of new algorithms and hardware in a fraction of the time required by traditional approaches. A screenshot of such an application is shown in Figure 4. For the quantum chemistry packages adopting this new design paradigm [GAMESS (Ames Laboratory), NWChem (Pacific Northwest National Laboratory) and MPQC (Sandia National Laboratories) thus far], many advantages have already been realized, including the abilities to leverage more efficient optimization components written by domain experts, transparently share low-level integral evaluation routines, more efficiently utilize high performance computers, and automatically tune applications for specific molecular systems and hardware environments.

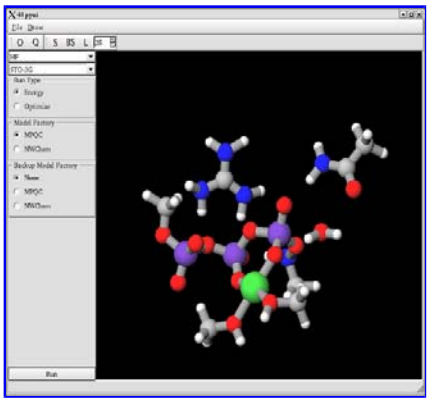

Figure 4. Screenshot of a graphical component front-end developed within the QCSAP.

The Future

Some of the ongoing work includes advances in component capabilities for massively parallel and heterogeneous architectures, runtime enforcement of behavioral semantics, additional expressibility for complex interactions between components, and parallel coupling. We provide brief highlights below; see [[3], [14]] for additional details.

Emerging HPC Environments. Scientists developing petascale computational science capabilities continue to face major challenges in effectively using emerging high-performance computing (HPC) architectures, which are characterized by large processor counts and increasing use of heterogeneous, specialized environments. We are thus developing new tools for CCA users to simplify and accelerate the development of true petascale applications on diverse hardware platforms. Our goals are that CCA users will be able to flexibly and dynamically express higher levels of parallelism,[15] transparently exploit specialized coprocessing resources, and support intelligent application-level responses to the hardware failures that are inevitable on systems of this scale. For example, we are working with a bioinformatics/proteomics application team to analyze data generated by mass spectrometers at PNNL.[16]

Software Quality and Verification. To help make the vision of interchangeable components a reality for scientific software, we are developing capabilities for the composition- and execution-time verification of interface semantics.[17] [18] Component interfaces, expressed separately from implementations, can be extended with semantic information to provide concise specifications that are both human-readable and interpreted by software. Unlike traditional verification techniques based either on post-execution comparisons with prior or analytical results or on algorithm-based fault tolerance techniques, this approach enables error detection closer to the point of failure. The result is improved testing, debugging, and runtime monitoring of software quality, thereby providing software developers with a powerful tool for catching errors early and ensuring correct software usage.

Computational Quality of Service. As computational science progresses toward ever more realistic multi-physics applications, no single research group can effectively select or tune all components of a given application, and no solution strategy can seamlessly span the entire spectrum efficiently. Common component interfaces enable easy access to suites of independently developed algorithms and implementations. The challenge then becomes how, during runtime, to make the best choices for reliability, accuracy, and performance. As motivated by simulations in combustion,[10] quantum chemistry,[19] fusion,[13] and accelerators,[20]
TASCS researchers are addressing this challenge by developing tools for Computational Quality of Service (CQoS), or the automatic selection and configuration of components to suit a particular computational purpose and environment.[21] The two main facets of CQoS tools are (1) measurement and analysis infrastructure and (2) control infrastructure for dynamic component replacement and domain-specific decision making.[22]

Parallel Data Redistribution and Parallel Remote Method Invocation. Parallel components raise questions about the semantics of method invocations and the mechanics of parallel data redistribution involving these components. Method invocations between parallel components are an opportunity to automate the data redistribution and translation semantics of the interaction between those components. The so-called MxN problem (where M processors associated with one component coordinate with N processors associated with another) arises often when multiple simulation components are joined in a single application. This allows an application to utilize a combination of task-based parallelism and domain decomposition to achieve integration, regardless of the scaling characteristics and resource constraints of the individual components. Support for this capability is being integrated into the Babel compiler.

Conclusion

The Common Component Architecture is a solid foundation for developing modular, maintainable high-performance simulations. Through support of the TASCS Center and collaborators, the surrounding ecosystem continues to flourish, and provides new functionality for taming the complexity of multi-physics, multi-scale, scalable applications.

Acknowledgment
This work was supported by the U.S. Department of Energy's Scientific Discovery through Advanced Computing (SciDAC) program, through the Office of Advanced Scientific Computing Research, Office of Science. The CCA Forum is a community involving participants for numerous DOE national laboratories, Universities, Companies, and other organizations.

[1] Szyperski, C. "Component Software: Beyond Object-Oriented Programming," *ACM Press*, New York, 1999.

[2] Allan, B. A., Armstrong, R., Bernholdt, D. E., Bertrand, F., Chiu, K., Dahlgren, T. L., Damevski, K., Elwasif, W. R., Epperly, T. G. W., Govindaraju, M., Katz, D. S., Kohl, J. A., Krishnan, M., Kumfert, G., Larson, J. W., Lefantzi, S., Lewis, M. J., Malony, A. D., McInnes, L. C., Nieplocha, J., Norris, B., Parker, S. G., Ray, J., Shende, S., Windus, T. L., Zhou, S. "A Component Architecture for High-Performance Scientific Computing," *Intl. J. High-Perf. Computing Appl.*, 2006, pp. 163-202.

[3] CCA Forum - http://cca-forum.org

[4] Cactus - http://www.cactuscode.org

[5] Earth System Modeling Framework - http://www.esmf.ucar.edu/

[6] Collins, N., Theurich, G., DeLuca, C., Suarez, M., Trayanov, A., Balaji, V., Li, P., Yang, W., Hill, C., da Silva, A. "Design and Implementation of Components in the Earth System Modeling Framework," *Intl. J. High-Perf. Computing Appl.*, 2005, pp. 341-350.

[7] CCA Specification - https://www.cca-forum.org/wiki/tiki-index.php?page=CCA+Specification

[8] Babel - http://www.llnl.gov/CASC/components/babel.html

[9] CCaffeine - http://www.cca-forum.org/ccafe/

[10] Najm, H. (PI), Computational Facility for Reacting Flow Science (CFRFS) - http://cfrfs.ca.sandia.gov/.

[11] SWIM - http://cswim.org

[12] CPES - http://www.cims.nyu.edu/cpes/

[13] FACETS - https://www.facetsproject.org/facets

[14] McInnes, L., Dahlgren, T., Nieplocha, J., Bernholdt, D., Allan, B., Armstrong, R., Chavarria, D., Elwasif, W., Gorton, I., Kenny, J., Krishan, M., Malony, A., Norris, B., Ray, J., Shende, S. "Research Initiatives for Plug-and-Play Scientific Computing," *Journal of Physics*: Conference Series 78 (2007), (available via http://www.iop.org/EJ/abstract/1742-6596/78/1/012046).

[15] Krishnan, M., Alexeev, Y., Windus, T., Nieplocha, J. "Multilevel Parallelism in Computational Chemistry using Common Component Architecture and Global Arrays," *Proceedings of SuperComputing, 2005*.

[16] High-Performance Mass Spectrometry Facility, Pacific Northwest National Laboratory - http://www.emsl.pnl.gov/capabs/hpmsf.shtml

[17] Dahlgren, T., Devanbu, P. "Improving Scientific Software Component Quality Through Assertions," *Proc. 2nd Int. Workshop on Software Engineering for High Performance Computing System Applications*, 2005, pp. 73-77, (available via http://csdl.ics.hawaii.edu/se-hpcs/papers/sehpcs-proceedings.pdf).

[18] Dahlgren, T. "Performance-Driven Interface Contract Enforcement for Scientific Components," *Proc. 10th Int. Symp. on Component-Based Software Engineering*, LNCS 4608, Springer-Verlag, 2007, pp. 157-172.

[19] Gordon, M. (PI), Chemistry Framework using the CCA - http://www.scidac.gov/matchem/better.html

[20] Spentzouris, P. (PI), SciDAC Community Petascale Project for Accelerator Science and Simulation (COMPASS) - https://compass.fnal.gov

[21] Norris, B., Ray, J., Armstrong, R., McInnes, L.,Bernholdt, D., Elwasif, W., Malony, A., Shende, S. "Computational Quality of Service for Scientific Components," *Proc. Int. Symp. on Component-Based Software Engineering*, 2004, Edinburgh, Scotland (available via ftp://info.mcs.anl.gov/pub/tech_reports/reports/P1131.pdf).

[22] McInnes, L., Ray, J., Armstrong, R., Dahlgren, T., Malony, A., Norris, B., Shende, S., Kenny, J., and Steensland, J. "Computational Quality of Service for Scientific CCA Applications: Composition, Substitution, and Reconfiguration," *Argonne National Laboratory preprint ANL/MCS-P1326-0206*, 2006, (available via ftp://info.mcs.anl.gov/pub/tech_reports/reports/P1326.pdf).

URL to article: http://www.ctwatch.org/quarterly/articles/2007/11/enabling-advanced-scientific-computing-software/