UCRL-CONF-222279

How the Common Component Architecture Advances Computational Science

Gary Kumfert, David E. Bernholdt, Thomas Epperly, James Kohl, Lois Curfman McInnes, Steven Parker, and Jaideep Ray

26 June 2006

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.

U.S. Department of Energy

Lawrence Livermore National Laboratory

How the Common Component Architecture Advances Computational Science

G Kumfert[1], **D E Bernholdt**[2], **T G W Epperly**[1], **J A Kohl**[2], **L C McInnes**[3], **S Parker**[4] **and J Ray**[5]

[1] Lawrence Livermore National Laboratory
[2] Oak Ridge National Laboratory
[3] Argonne National Laboratory
[4] University of Utah
[5] Sandia National Laboratory

E-mail: kumfert@llnl.gov

Abstract. Computational chemists are using Common Component Architecture (CCA) technology to increase the parallel scalability of their application ten-fold. Combustion researchers are publishing science faster because the CCA manages software complexity for them. Both the solver and meshing communities in SciDAC are converging on community interface standards as a direct response to the novel level of interoperability that CCA presents. Yet, there is much more to do before component technology becomes mainstream computational science. This paper highlights the impact that the CCA has made on scientific applications, conveys some lessons learned from five years of the SciDAC program, and previews where applications could go with the additional capabilities that the CCA has planned for SciDAC 2.

1. Introduction

Component technology exists because people are not scalable. Throughout the short history of software development, it has always been the case that (a) human beings write imperfect software, and (b) they need to produce ever-increasing amounts of it, nonetheless. This situation is the essence of the perennial "software crisis." Software technologies such as assemblers, compilers, structured programming, object-oriented programming (OOP), and now component-based software engineering (CBSE) have been created in response to this need, expanding by an order of magnitude or more the scale of software producible. Each technology addresses this issue by raising the levels of abstraction, enforcing more programming structure, generating more code internally per line of developer code, and ultimately protecting developers from their own human limitations. Unfortunately all of these technologies eventually succumb to their own inherent scaling limit; human foibles are no longer effectively mitigated. The resulting defects dominate the system. The progression of programming technologies augments — but does not replace — their predecessors. Each tool solves a particular problem, and the choice of tools depends on the size and nature of the programming task.

Component technology is most effective when the target software has achieved a level of complexity that exceeds the possible comprehension of a single human mind, even a domain expert with ample access and time. Literature often uses the term *enterprise software*, but this term suffers from multiple deficits. Although it is generally understood to be software of sufficient capability and merit to be applicable beyond a single individual, team, or department, many interpret it to apply only to business processes across the corporate enterprise. The term is also unsatisfying because many in computational science

have observed long-lived applications that do not leave a single department, but have accreted so much additional complexity over decades of use, that they too are prime candidates for componentization.

When software is small enough, or there is a guru talented enough to understand the complete code, the incentives for component technology are less obvious but no less compelling. The most frequently cited motivation for components, code reuse, is not the most convincing argument in practice. Stronger arguments can be made, but are specific to the distinct needs of corporate and scientific computing. For industry, time to market is the key consideration. Because components are loosely coupled entities, the reduction in intra-dependencies allows more parallelism in the development process. Thus companies can have more programmers productively working at the same time and can shorten the critical development path. Scientific computing has a completely different nature; rather than a race to a single release, scientific codes need to nimbly adapt through decades of change. Change happens externally through new hardware generations and incremental updates of third-party software as well as internally as scientific understanding evolves, new algorithms present themselves, and new questions are explored in pursuit of science. Perhaps the most compelling argument for component technology in scientific computation is maximizing adaptability and maintaining correctness in the face of such constant change.

To demonstrate the impact that the Common Component Architecture (CCA) has on science, this paper surveys the use of the CCA in the high-performance computational community. We present background information specific to the CCA in Section 2. The bulk of the paper is in Section 3, which enumerates six different modes in which our work has affected science; each subsection names several representative projects across a broad spectrum of applications and disciplines. In Section 4, we summarize experiences from the five years of SciDAC and tie this into our technology plan for the next five years of SciDAC2. Finally, we close in Section 5 with our forecast of what else is needed to bring component technology into mainstream computational science.

2. Background

Component-based software engineering (CBSE) is a field of study that seeks to improve the quality (flexibility, maintainability, reliability) of software systems while reducing costs (production and time-to-market). Inspired by modular and interchangeable hardware components in electronics, CBSE attempts to replicate this effect in software through a mix of tools, framework infrastructure, and coding methodologies.

The earliest and perhaps best-known instance of software assembled from prefabricated components is the system of pipes and filters built into the UNIX operating system. This system was invented by M. Douglas McIlroy, who in 1968 first argued for the industrialized manufacture and use of componentized software [1]. The modern concept of software components motivated the creation of the Objective-C programming language [2]. But it was ultimately the failings of the object-oriented paradigm at enterprise scales that led to the development of component technology [3]. The most frequently cited failings of OOP include the implicit assumptions that all software entities to be integrated are written in the same programming language and are amenable to inheritance and aggregation [4].

2.1. Challenges Unique to Scientific Software Componentry

Scientific computing fundamentally requires maximal performance of the underlying hardware; whether it be laptop, workstation, commodity cluster, or leadership-class machine. A minimally viable component system for scientific computation must support Fortran, complex arithmetic, and multidimensional dynamically allocated arrays (preferably with arbitrary strides). It must provide a tenable migration strategy for existing software assets, incur minimal runtime overhead, and be portable to most of the machines on the Top 500 list [5]. Of the dominating component systems of the corporate world, CORBA/CCM [6], COM/COM+ [7], J2EE/EJB [8], and .Net [9], not one satisfies all these criteria for scientific computation [10]. Developing and delivering a suitable component system for this domain has been the mission of the Common Component Architecture Forum (CCA Forum) since its inception

1998 and SciDAC's Center for Component Technology for Terascale Simulation Science (CCTTSS), which supported the majority of the CCA Forum participants, from 2001–2006.

The CCA is itself a modular stack of technologies. At its base is a language for specifying generic software interfaces called the Scientific Interface Definition Language (SIDL). The Babel tool [11] reads SIDL files and generates wrapper code that supports a uniform object-oriented model across six languages in a single address space. The CCA specification is written in SIDL and specifies how components interact with each other and the underlying framework. There are several CCA compliant frameworks available, but the SciDAC effort supports three core implementations of the CCA specification. Ccaffeine [12], emphasizes an enhanced SPMD-style programming model and also supports a native C++ interface. XCAT [13, 14] focuses on distributed Web Services-style programs. SCIRun2 [15] has bridging technologies between CCA, CORBA, VTK, and shared-memory dataflow models.

2.2. The SIDL/Babel Technology

The key to making software interoperable is a consistent type system. This includes basic types, such as integers and strings, compound types such as arrays, and user-defined types such as structs, classes, or derived types. Although every programming language necessarily has an internally consistent type system, the union of these languages is a mess. Every language has its own character string type. Arrays can be row-major, column-major, or 1 dimensional only. Dynamically allocated might be reference counted, garbage collected, the programmer's burden or nonexistent. Error messages could be stack-unwinding exception classes that foreign languages normally cannot intercept. We have defined a trans-language type system that — with the assistance of generated code, runtime libraries, and programmer discipline — is completely consistent across C, C++, Fortran 77, Fortran 90, Java, and Python. Babel is the software package that performs the code generation and provides the supporting runtime libraries. SIDL is the input language that drives Babel's code generation.

Our Scientific Interface Definition Language (SIDL) is one of a family of interface languages that includes CORBA IDL and COM IDL.[1] These interface languages exist for users to describe their own types and the signatures of subroutines bound to those types. Distinguishing characteristics of SIDL are that it is a much smaller and straighforward language than either CORBA's or COM's IDLs; designed for users that are general computational scientists and not necessarily full-time expert programmers. SIDL uniquely counts among its built-in types full Fortran 90-featured arrays, C/Fortran 77-style *raw arrays*, and complex numbers. SIDL continues to add new constructs in response to customer needs and is itself the subject of external research. Recent graduate theses have also investigated adding parallel data types and directives [16, 17] and semantic constraints and enforcement [18].

Babel has two main parts: a code generator, and a runtime library. The code generator reads SIDL input files and generates wrapper code that is far more sophisticated, robust, consistent, and portable than what people would ever write by hand. Newcomers are often taken aback by the volume of code generated by Babel. Interoperability is a difficult problem dominated by hundreds of arcane little details and it takes an appreciable amount of code to properly handle them all. To completely encapsulate language dependencies so a caller never knows (nor cares) what language they are calling into, the type system had to be fully object-oriented. The generated wrappers, therefore, have a complete virtual function dispatch system embedded inside. The runtime library includes common base classes, exception classes, and language specific support for a consistent type system regardless of implementation language.

The distinguishing characteristics of the Babel system are that languages are mixed in a single address space with no messaging or interpreted languages brokering the interoperability. This is technically hard to accomplish, but gives outstanding performance. In the worst case, Babel is 25% faster than

[1] Not to be confused with the Interactive Data Language, which is completely unrelated, but goes by the same acronym. The Interactive Data Language is a full programming language of VAX/VMS/Fortran heritage that is used for interactive data and image processing.

Table 1. CCA Customers by Application Domain

Domain	Project	POC	Section	Page
accelerator beam dynamics	Beam-SBIR	Douglas Dechow, Tech-X Corp.	4	10
cell biology	VMCS	Harold Trease, PNNL	3.3	8
chemistry	NWChem	Theresa Windus, PNNL	3.1 & 3.2	5, 7
chemistry	MPQC	Curtis Janssen, SNL	3.2	7
chemistry	GAMESS-CCA	Masha Sosonkina, Ames Lab	3.3	8
climate	ESMF	Nancy Collins, NCAR	3.6	10
combusion	CFRFS	Jaideep Ray, SNL	3.1	4
electron effects	CMEE	Peter Stoltz, Tech-X Corp.	3.4	8
frameworks	MOCCA	Vaidy Sunderam, Emory Univ.	3.5	9
fusion	DFC	Nanbor Wang, Tech-X Corp.	3.2	7
fusion	FMCFM	Johan Carlsson, Tech-X Corp.	3.4	8
geomagnetics	—	Shujia Zhou, NASA	3.1	6
materials	PSI	David Jefferson, LLNL	3.4	8
meshing	TSTT	Lori Diachin, LLNL	3.3	7
nuclear power plant	—	M. Díaz, Univ. of Málaga	3.2	7
performance	TAU	Sameer Shende, Univ. Oregon	3.1	6
radio astronomy	eMiriad	Atholl Kemball, UIUC	3.2	7
solvers	hypre	Jeff Painter, LLNL	3.4	8
solvers	TOPS	Barry Smith, ANL	3.3	7
sourcecode refactoring	CASC	Dan Quinlan, LLNL	3.4	9
sparse linear algebra	SPARSKIT-CCA	Masha Sosonkina, Ames Lab	3.1	6
subsurface transport	PSE Compiler	Jan Prins, UNC Chapel Hill	3.1	6

its competitors, and the margin only increases when characteristically large scientfic data is exchanged between the layers. Babel won a R&D 100 award in 2006 in recognition of its unmatched peformance.

3. Impact of CCA on Science

Different scientific application domains and teams have diverse technical needs and cultures. It should be no surprise, therefore, that the computational science community's response to the CCA has been diverse. Table 1 contains a representative, but far from exhaustive, list of projects where we have observed the CCA's impact. These observations are grouped in Sections 3.1–3.6 according to six modes of adopting and employing CCA technology.

3.1. Maximizing Flexibility in New Codes

This section discusses applications that have adopted the CCA and thus now employ CCA technology as they develop new code. Their primary motivation is not reuse or sharing code, but rather increased flexibility in the process of scientific exploration.

Combustion. The Computational Facility for Reacting Flow Science (CFRFS) [19] has used the CCA to develop a toolkit for simulating and analyzing high-fidelity reacting flows with detailed chemistry. The componentized form has enabled domain experts to develop solutions independently and to tolerate a great deal of developer turn-over. CFRFS researchers were first in the field to employ high-order (fourth-order and higher) discretization approaches [20] and extended-stability explicit integrators [21]

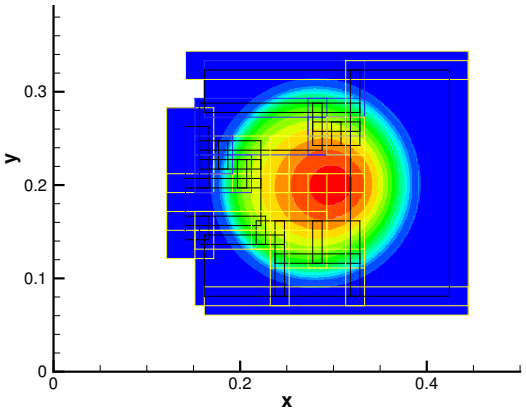

Figure 1. OH distribution from an advective-diffusive-reactive simulation using fourth-order spatial discretization and a Runge-Kutta-Chebyshev integrator on a 4-level mesh hierarchy. Solutions on 25μm (yellow borders) and 12.5μm (black borders) patches are shown.

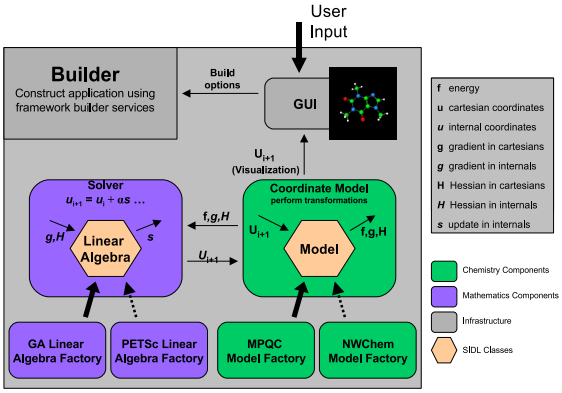

Figure 2. The high-level component architecture for NWChem and MPQC courtesy of Curtis Janssen and Joseph P. Kenny, Sandia National Laboratories

on block-structured adaptive meshes. Figure 1 shows an OH distribution from an advective-diffusive-reactive simulation of igniting hotspots in a stoichiometric H_2-Air mixture on a 4-level block-structured adaptive mesh. Using the CFRFS Toolkit, a Runge-Kutta-Chebyshev algorithm [22] was employed with a fourth-order spatial discretization approach [20]. Further, their automatic detection and evolution of systems on low-dimensional manifolds using Computational Singular Perturbation [23, 24] hold significant potential for reducing the computational expense of integrating stiff chemical systems. The CFRFS toolkit was also used to explore runtime performance optimization by dynamically detecting and replacing components with sub-par performance [25, 26].

This project is particularly interesting because in addition to pursuing combustion research, CFRFS researchers have also quantitatively evaluated the merits of componentization [27]. The toolkit is a collection of approximately 60 components, covering a range of functionality for physico-chemical and transport models, numerical schemes (integrators, nonlinear solvers, etc.), as well as parallel meshes and domain-decomposed data managers. The vast majority of these components are small, with less than 1000 lines of code. The interfaces to the components usually contain less than 10 functions, yet even such simple interfaces enjoy multiple uses. In addition to internally developed assets, the CFRFS toolkit has componentized wrappers to external packages, such as a time integrator (CVODE [28]), a parallel linear solver (hypre [29]), block-structured adaptive meshes (GrACE [30]), and several legacy physico-chemical models from Sandia. Much of the code does not use Babel but an earlier C++-only interface to the Ccaffeine framework, now called *Ccaffeine classic*. Ccaffeine itself provides the bridging technology to connect classic components to Babel components.

Chemistry. A Multiple Component/Multiple Data (MCMD) approach to parallelism, based on CCA, was implemented using NWChem [31] and a modified variant of Global Arrays [32] to implement numerical Hessian calculations using three levels of parallelism. The CCA driver component, which had responsibility for the overall computation, instantiated several NWChem components over different subgroups of nodes. Each of these NWChem components then performed multiple parallel energy computations on its subset of nodes to determine the gradient, providing a multi-level parallel application. Using this approach for a simple five-water cluster, an order of magnitude improvement in time to solution over the SPMD approach was observed for 256 processors [33]. This same type of approach

will be applied to other chemistry algorithms such as simulated annealing, vibrational self-consistent field and Monte Carlo methods.

Subsurface Transport. Researchers at the Center for Advanced Study of the Environment (CASE), University of North Carolina, Chapel Hill, are applying CCA tools and technology in their problem solving environment (PSE) for subsurface flow and transport phenomena [34]. Using LaTeX as a specification language for sets of differential equations, their "PSE compiler" translates a symbolic flow/transport problem into a component-based simulation. A variety of externally developed solver, integrator, and utility components have been identified and coordinated through a shared knowledge base to satisfy the needs of the simulation along with a simple Babel-wrapped component description of the model itself. The resulting component-based program is then submitted for parallel execution using the Ccaffeine CCA framework.

This team reports several benefits of the CCA approach. The component-based paradigm enables isolated unit testing of various solver/integrator components and provides a flexible platform for experimentation, where users can swap key portions of the component network without requiring full knowledge of the overall algorithms and internal organization. The high-level structure of the component-based representation encourages an intuitive understanding of the overall solver organization for users, versus traditional monolithic codes. The well-defined SIDL component interfaces also alleviate complexity in the design of the PSE compiler, abstracting the functional relationships among components and hiding specific implementation details that are not directly relevant at the PSE compiler level.

Geomagnetics. XCAT is the CCA framework of choice for long-haul distributed computing. In a feasibiliy study [35], researchers used XCAT to create an ensemble of MoSST (Modular, Scalable, Self-consistent, Three-dimensional) [36, 37] core dynamics models. They linked the federation via XCAT's built-in Grid support across a 10G network, and they employed Jython, a Java implementation of Python, to provide a scripted front-end for ease of use.

Performance Monitoring. TAU [38] is a robust and portable measurement interface and system for software performance evaluation. Using SIDL to describe TAU's measurement API, full support was enabled across applications written in Fortran, C++, C, Java, and Python. Without such support, the API for each new target language would be independently developed and maintained. Such a complex task becomes even more difficult given the ongoing sequence of extensions evolving in the TAU measurement API. Babel helps the TAU team focus on improving the quality of performance measurement and analysis tools, instead of dealing with low-level language compatibility. 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.

Sparse Linear Algebra. Sparskit [39] is a basic toolkit (written in F77) for sparse linear algebra, with a significant portion (80%) now componentized for the CCA toolkit. The Sparskit components are also being integrated into the Terascale Optimal PDE Simulation (TOPS) [40] center's solver component [41]. New algebraic multilevel methods (in C), from the Itsol [42] package — a library of iterative solvers for general sparse linear systems of equations, an extension of Sparskit — have now been merged as components into the CCA Toolkit. TAU's component-based performance analysis tools have been applied to the Sparskit linear algebra components. This study found that components of a fine granularity, like those in Sparskit, still execute with acceptable overheads rates of less than 3.4% in common application usage.

3.2. Combining Legacy Codes

The computational science community has huge existing investments in a broad assortment of physics, chemistry, numerical, system, and visualization software. Often, one can realize a scientific breakthrough

by combining best-in-class technologies from different disciplines into an integrated application. This very simple concept is often difficult to achieve in practice due to codes using different programming languages, data models, units of measurement, or differing standards. The CCA provides the tools to wrap legacy libraries as components with relatively simple interfaces, thereby enabling integrated applications using best-in-class technologies.

Quantum Chemistry. The NWChem [31] and MPQC [43] teams used the CCA to combine their quantum chemistry models with the TAO [44] optimization package, PETSc [45], and Global Arrays [32] to improve the accuracy and performance of their application. Choosing a coarse-grained componentization with an architecture shown in Figure 2 [46, 47], they defined and shared a common SIDL interface to provide the energy, gradient, and Hessian to the optimization component. By decoupling the optimization algorithms from the quantum chemistry calculations, NWChem and MPQC were able to incorporate optimization algorithms developed by experts, which led to a net reduction in the number of iterations required for overall solution [47]. In addition, these groups are now poised to take advantage of new advances in optimization technology as they become available.

Nuclear Plant Simulation. Researchers from the University of Málaga in Spain are using the CCA along with Real-Time CORBA (RT-CORBA) [48] to create a nuclear power plant simulator to train operators [49]. They chose to use RT-CORBA for the user interface and data logging subsystems, where predictable response time is required, along with the CCA for the simulator kernel, where high performance and support for Fortran are needed. This team started with a software system where data was shared among software subsystems using global variables. Using the CCA, they created a loosely coupled simulator kernel, where each component has a well defined interface indicating what data it requires and provides. During the configuration phase, the simulator kernel defines a communication schedule to satisfy the data dependencies among models. By componentizing the simulator, they lowered their development costs and produced a more flexible simulator.

Fusion. A team at the Tech-X Corporation is working on a distributed components project to integrate and connect components from different CCA frameworks. This work will enable scientists to utilize distributed network resources for data storage or post processing, to connect to existing distributed services, and to compose loosely coupled applications where each component is running on its optimal parallel architecture. This project is working with a componentized, legacy fusion code produced by an ORNL Laboratory Directed Research and Development project, AORSA [50].

Radio Astronomy. The eMiriad project at UIUC is developing a domain-specific component framework based on Babel to integrate several legacy libraries to make a radio astronomy application. This project will make a variety of tools available to scientists through common interfaces. In particular, they are integrating AIPS, MIRIAD, and AIPS++, which together represent approximately 480 FTE-years of effort [51]. They chose Babel as their middleware because it is particularly well suited to their domain, radio astronomy imaging. The support for multi-dimensional arrays, Fortran, good interoperability with parallel computing, and the quality of peer-to-peer language bindings were leading factors in choosing Babel. Babel's language interoperability capabilities enable developers to work in their most effective programming language and provide a general scripting interface for the integrated system using Python.

3.3. Common Interfaces

The new level of interoperability that component technology supports has also spurred renewed interest in developing community-based common interfaces. Initially, participants often underestimate the effort and commitment required for a community to gather and agree on a precise set of terms, let alone a set of interfaces. A discipline-specific interface that is generated and agreed to by a community is a vital intellectual product in its own right [51].

Solvers and Meshes. The two largest SciDAC teams active in producing common SIDL interfaces are the Terascale Optimal PDE Simulations (TOPS) [40] project and Terascale Simulation Tools and Technologies (TSTT) Center [52], which focus on solvers and meshing, respectively. It is particularly

interesting to note that these two applications represent opposing extremes in natural problem granularity. Solvers tend to be large-grained with plenty of work per method invocation to completely swamp any component overhead [53, 47, 54, 55]. In contrast, meshing naturally has a fine-grained interface where not much data resides behind a single node, edge, or zone, because operations are done iterating across many of them. However, experiments demonstrate that only a moderate granularity of access is needed to amortize overhead for meshing components [55].

PNNL scientists are using TSTT tools to build the Virtual Microbial Cell Simulation (VMCS) to solve DOE heavy metal waste bioremediation problems. The VMCS is a general biological application that couples individual microbes, each modeled as its own genome-scale metabolic network, into a larger, self-organizing spatial network. The communication between the organisms is provided by multi-dimensional flow and transport models. TSTT mesh generation, mesh quality improvement, and discretization tools developed at different sites, and written in different languages, are used in concert through the SIDL-based TSTT interfaces. The VCMS has been used to study the flocculation behavior of communities of Shewanella microbes in oxygen rich environments. These simulations confirmed that there is an oxygen gradient from the edges of the floc into the center and provided new insight into the behavior of these microbes.

Chemistry. Perhaps the best benefit of developing common interfaces is that the value increases as the community grows. For example, the interfaces developed by our NWChem and MPQC customers have recently also been employed by others in the creation of GAMESS-CCA [56].

3.4. CCA a la Carte

In addition to many applications that use full CCA componentry, there are more that employ specific technologies from the CCA arsenal. One of the more visible technologies is the Babel interoperability tool and its constituent Scientific Interface Definition Language (SIDL).

Electron Effects in Heavy Ion Fusion Accelerators. The Computational Models for Electron Effects (CMEE) [57, 58] takes widely-used physics routines for modeling electron effects like gas ionization and secondary electron emission from metals and uses Babel to make them widely available. The resulting code is used in applications such as accelerator physics and plasma drives for satellites. In addition to having legacy codes in Fortran 77, they also report integrating new codes in Fortran 90, C, and Python.

Before incorporating Babel, this project had used combinations of Pyfort [59] and SWIG [60] or f2py [61] and SWIG, which reportedly gave them 90% of what they wanted. However, a new customer (U.S. Air Force) added the requirement of Java interfaces. Rather than discard their substantial investment in Python-based string parsing code, they replaced all other point-to-point language tools with SIDL/Babel. This is the first known case of a Babel customer having a critical need for Java calling Python. They report taking a half-person day to demonstrate Java calling Python in their own code.

Material Science. The Petascale Simulation Initiative (PSI) [62] is investigating combinations of whole SPMD programs as distributed federations within a single petascale machine. They are developing this technology to perform multi-scale material science calculations, where an established continuum engineering code is connected to a farm of sub-scale crystal plasticity and dislocation dynamic simulations. Not only do they use Babel to connect their infrastructure (C++), engineering code (ANSI C), and sub-scale calculations (Fortran), they also invest their own funding to accelerate the development of remote method invocation (RMI) in Babel [63].

Fusion. The Framework for Modernization and Componentization of Fusion Modules (FMCFM) is developing SIDL interfaces for legacy codes, particularly in the subareas of equilibrium and transport. Based on the Fortran-centric APIs from the European Integrated Tokamak Modeling Task Force (ITM-TF), this project is actively brokering a compromise position that involves SIDL as a secondary interface to the native Fortran 90. This group is working technically with the Babel team to pursue the inclusion of structs (derived types) and optional arguments in a future release.

Solvers. The *hypre* library of scalable solvers and preconditioners [29] was the first tester of

SIDL/Babel. Written mostly in ANSI C, the hypre team's original intention was to throw away the four hand-written, non-portable, partial Fortran wrappers and use Babel as the binding for all languages other than the core C interface. After years of experience, performance studies showing the Babel overhead to be unmeasurable in large parallel jobs [54], and the benefits of polymorphism, the long-range plan has shifted to Babel being the *only* interface published to customers.

Computer-Assisted Sourcecode Refactoring. Researchers at LLNL are developing a methodology to analyze and refactor large amounts of sourcecode. Applications include finding/breaking cyclic dependencies, removing blacklisted programming constructs, automatic wrapping in Babel, and possibly even semi-automated componentization [64]. The CCA itself is developing a simple but effective scripted approach to automated conversion from a language-specific CCA-Lite form to full Babel/SIDL based on Chasm [65, 66]. By comparison, the LLNL project is larger and more general. It employs the Rose Compiler Framework [67], the Eclipse IDE [68], and the VizzAnalyzer [69, 70] software visualization tool to manipulate the entire C, C++, or Fortran source in memory with as much detail as a commercial compiler. This project uses Babel to connect Rose (implemented in C++) to Eclipse and VizzAnalyzer, which are both implemented in Java. This project also motivated and developed the *back-door initializer* feature in Babel to wrap native objects in a temporary Babel veneer.

3.5. Framework Interoperability

Numerous CCA projects have focused on component interoperability, which can be classified into two main forms: internal interoperability and external interoperability. Internal interoperability focuses both on ensuring that a CCA component will work in any of the disparate CCA frameworks, and that multiple CCA frameworks can coordinate with each other if necessary. The CCA has ongoing efforts toward these goals, but several CCA members have focused on the stronger external form of interoperability. Industry standard component frameworks, such as CORBA, Microsoft COM [71], and Enterprise Java Beans [72]), component-based software libraries (such as VTK [73]), workflow systems (such as Kepler [74, 75]), or Problem-Solving Environments (such as SCIRun [76, 77]) may all have properties or components that are desirable to use in a scientific application.

Consequently, several systems have created mechanisms for interoperating with other component-based or grid-based software systems. Each of these systems attempts to provide the ability for a computational scientist to use the right tool for the right job, a goal motivated by the needs of scientific users who have existing software that is not implemented using the CCA.

SCIRun2 [15] focuses on enabling multiple component models to cooperatively coexist. The primary innovative design feature of SCIRun2 is a meta-component model that facilitates integration of components from disparate models. In the same way that components plug into the CCA or other component-based systems, SCIRun2 allows entire component models to be incorporated dynamically. Through this capability, SCIRun2 facilitates the coupling of multiple component models, each of which can bring together a variety of components. Researchers are utilizing this feature to enable the coupling of single-address-space components based on Babel, components from SCIRun, as well as CCA components that use the SCIRun2 distributed computing infrastructure. Special components, called bridges, facilitate interactions between components belonging to different models. These bridges can be automatically or semi-automatically generated.

Legion CCA [78] also seeks this type of interoperability by building on common design features and providing mechanisms to bridge between the specific details of interfaces. In particular, this software supports CCA over Legion. Programmers are able to specify CCA components in SIDL and run them through a Babel-like compiler to generate back-end code for Legion components. These components run within a new Legion CCA framework, implemented on top of a combination of existing and new Legion-based runtime mechanisms.

MOCCA [79] is a CCA-compliant framework implemented on top of the H2O distributed metacomputing framework. This work is part of a broader program of computer science research into distributed computing frameworks under the Harness project. H2O is a lightweight, pluggable

experimental infrastructure for building *personal grid* environments, as an alternative to the system-oriented Globus grid environment. H2O is written in Java and uses the RMIX multiprotocol communication library to communicate between H2O instances. H2O accommodates a variety of programming models, including PVM, MPI, OGSA web services, JRMP, SOAP, and RPC.

3.6. Intellectual Impact

Up to this point, we have focused on cases in which applications are working directly with the tools and environment that most users think of as "the CCA." However, we also increasingly observe groups within the computational science community incorporating the *ideas* of CBSE and the CCA into their software, providing their own implementations of component concepts rather than using CCA tools directly. We briefly highlight two examples.

Climate. The Earth System Modeling Framework (ESMF) [80, 81, 82] is an effort to develop a standard framework for applications in climate and weather modeling. The ESMF provides a substantial infrastructure of data structures and commonly used utilities. Higher-level application-specific functionality is cast in the form of user-provided software components, which employ ESMF infrastructure and are coordinated in their execution by ESMF superstructure. Though more restricted, the ESMF's component model draws on ideas and even terminology of the CCA. Interoperability between ESMF and CCA frameworks has been demonstrated [83, 84], and a closer linkage between the two frameworks has been discussed as a possible collaboration.

Astrophysics. The Terascale Supernova Initiative (TSI) [85], which focuses on computational modeling of core-collapse supernovae, has an aggressive software development agenda in order to satisfy their scientific goals, including both increasing the fidelity of the models of individual physical phenomena (such as increasing the dimensionality of models) as well as introducing and refining their couplings to other aspects of the physics. While not using the CCA tools directly, the team's new code developments are increasingly incorporating component concepts into their design, and realizing some of the same benefits as CCA users [86].

4. Future Directions of the CCA

Many new capabilities that CCA researchers will explore involve capitalizing on the adaptivity of components. For example, we have recently begun developing component capabilities for computational quality of service, heterogeneous and hybrid computing architectures, and advanced software verification, along with a richer suite of components in the CCA toolkit.

By automating the selection and configuration of components to suit computational conditions imposed by a simulation and its operating environment, we are developing infrastructure for Computational Quality of Service (CQoS) of CCA components [87, 88, 89, 90, 91, 92]. Motivated by collaborations in modeling accelerators, combustion, quantum chemistry, and fusion, and in collaboration with PERC [93], TOPS [40], and TSTT [52] researchers, we are developing a system that helps application scientists make suitable compromises among performance, accuracy, mathematical consistency, and reliability. A specific motivating example is the Synergia beam dynamics application [94] for high-energy accelerators, to which CCA components are being introduced in a new project [95]. CQoS work here focuses on the automation of appropriate choices for algorithms and parameters of TOPS linear solver components [41].

While the first SciDAC initiative existed during a period of relative homogeneity among processor architectures, exotic multi-core and hybrid cores are becoming increasingly common in high-performance computing. Hybrid computing couples field-programmable gate arrays (FPGAs), floating-point accelerators, vector processors, and other specialized hardware to traditional compute nodes. We are extending CCA component technology to manage interactions between special-purpose and general-purpose code in these heterogeneous environments.

Specification and dynamic enforcement of interface semantics are an important emerging approach to improving software quality. Whereas industry is doing this model-based software engineering

with XML pre-processors, we are adapting SIDL to specify semantics constraints directly in interface descriptions [96, 97]. This work will develop a powerful new feature for users to verify the correct use of third-party software.

5. Conclusion

If component technology is so effective, why isn't it already in the scientific computing mainstream? The answer is: time. Ultimately, programming is a human activity. People need time to explore new concepts before they can apply them effectively in their work. The fact that the strongest examples of the CCA's impact on scientific research come from our longest running collaborations is no accident. When one considers that more than thirty years were required for object-oriented programming to transition from first implementation (Simula67) to mainstream (ISO C++ standard was formally accepted in 1998), the results achieved by the CCA Forum in the first five years of SciDAC are remarkable.

There is still much work to do and many technical challenges to overcome. Because scientific computing spans so many computational architectures, has such demanding performance requirements, and requires massive parallelism, the work of the CCA Forum is far from complete. Nevertheless, this paper demonstrates that CCA component technology works, the approach is sound, and its impact on science continues to compound.

Acknowledgments

This work was funded by the U.S. Department of Energy/Office of Science's SciDAC program.

The CCA has been under development since 1998 by the CCA Forum and represents the contributions of many people, all of whom we gratefully acknowledge. We also thank our collaborators outside the CCA Forum, especially the researchers listed in Table 1, who took the time to contribute (and proof-read) their stories of how the CCA has affected their work.

Research at Argonne National Laboratory was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

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

Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the US Dept. of Energy under contract DE-AC-05-00OR22725.

Research at Sandia National Laboratories was supported by the US Department of Energy (DOE), Office of Basic Energy Sciences (BES), SciDAC Computational Chemistry Program. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

Research at the University of Utah is also sponsored by the National Science Foundation under contract ACI0113829, and the DOE ASC Program.

References

[1] M. Douglas McIlroy. Mass produced software components. In P. Naur and B. Randell, editors, *Software engineering: Report on a conference sponsored by the NATO Science Committee*, pages 128–155, Garmisch, Germany, October 1968. NATO Scientific Affairs Division.

[2] Brad J. Cox and Andrew J. Novobilski. *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, 2nd edition, 1991.

[3] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley/ACM Press, 1998.

[4] Douglas C. Schmidt and Steve Vinoski. The CORBA component model: Part1, evolving towards component middleware. *C/C++ User Journal*, January 2004.

[5] Top 500 list. `http://www.top500.org`.

[6] Object Management Group. *CORBA Component Model Specification*, 4.0 edition, April 2006. formal/2006-04-01.

[7] David Iseminger, editor. *COM+ Developer's Reference Library*. Windows Programming Reference Series. Microsoft Press, 2000.

[8] Enterprise JavaBeans Website. http://java.sun.com/products/ejb.

[9] Microsoft .NET homepage. http://www.microsoft.com/net.

[10] Rob Armstrong, Gary Kumfert, Lois Curfman McInnes, Steven Parker, Ben Allan, Matt Sottile, Thomas Epperly, and Tamara Dahlgren. The CCA component model for high-performance computing. *Intl. J. of Concurrency and Comput.: Practice and Experience*, 18(2), 2006.

[11] Tamara Dahlgren, Thomas Epperly, Gary Kumfert, and James Leek. *Babel User's Guide*. CASC, Lawrence Livermore National Laboratory, Livermore, CA, babel-0.99.0 edition, 2006.

[12] Benjamin A. Allan and Rob Armstrong. Ccaffeine framework: Composing and debugging applications interactively and running them statically, June 2005.

[13] The XCAT project. http://www.extreme.indiana.edu/xcat.

[14] Michael J. Lewis and Madhusudhan Govindaraju. The XCAT project. http://grid.cs.binghamton.edu/projects/xcat.html.

[15] K. Zhang, K. Damevski, V. Venkatachalapathy, and S.G. Parker. SCIRun2: A CCA framework for high performance computing. In *Proceedings of The 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, April 2004.

[16] K. Damevski. Parallel component interaction with an interface language compiler. Master's thesis, University of Utah, 2003.

[17] Felipe Bertrand. *Data Redistribution and Remote Method Invocation in Parallel Component Architectures*. PhD thesis, Indiana University, 2005.

[18] Tamara L. Dahlgren. *Adaptive Enforcement of Component Interface Assertions*. PhD thesis, University of California at Davis, One Shields Avenue, Davis, CA, 945616, In Progress. Also available as Lawrence Livermore National Laboratory Technical Report, UCRL-TH-221328-DRAFT.

[19] SciDAC Computational Facility for Reacting Flow Science (CFRFS). http://www.ca.sandia.gov/cfrfs.

[20] J. Ray, C. A. Kennedy, S. Lefantzi, and H. N. Najm. Using high-order methods on adaptively refined block-structured meshes–discretizations, interpolations, and filters. *SIAM Journal on Scientific Computing*, 2006. in review.

[21] S. Lefantzi, J., C. A. Kennedy, and H. N. Najm. A component-based toolkit for reacting flows with high order spatial discretizations on structured adaptively refined meshes. *Progress in Computational Fluid Dynamics: An International Journal*, 5(6):298–315, 2005.

[22] J. G. Verwer, B. P. Sommeijer, and W. Hundsdorfer. RKC time-stepping for advection-diffusion-reaction problems. *J. Comput. Phys.*, 201(1):61–79, 2004.

[23] J. C. Lee, H. N. Najm, S. Lefantzi, J. Ray, M. Frenklach, M. Valorani, and D. Goussis. A CSP and tabulation based adaptive chemistry model. *Combustion Theory and Modeling*, 2006. in press.

[24] M. Valorani, F. Creta, D. A. Goussis, J. C. Lee, and H. N. Najm. An automatic procedure for the simplification of chemical kinetics mechanisms based on CSP. *Combustion and Flame*, 2006. in press.

[25] Nicholas Dale Trebon. Performance measurement and modeling of component applications in a high performance computing environment. Master's thesis, University of Oregon, June 2005.

[26] J. Ray, N. Trebon, S. Shende, R. C. Armstrong, and A. Malony. Performance measurement and modeling of component applications in a high performance computing environment : A case study. Technical Report SAND2003-8631, Sandia National Laboratories, November 2003. Accepted, International Parallel and Distributed Computing Symposium, 2004, Santa Fe, NM.

[27] B. A. Allan, S. Lefantzi, and Jaideep Ray. The scalability impact of a component-based software engineering framework on a growing SAMR toolkit: A case study. In *Proccedings of Parallel Computational Fluid Dynamics*. Elsevier/North Holland, May 2005.

[28] S. D. Cohen and A. C. Hindmarsh. CVODE, a stiff/nonstiff ODE solver in C. *Computers in Physics*, 10(2):138–143, 1996.

[29] R. D. Falgout and U. M. Yang. hypre: A library of high performance preconditioners, in computational science. In P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, and A. G. Hoekstra, editors, *Lecture Notes in Computer Science*, volume 2331, pages 632–641. Springer-Verlag, 2002.

[30] GrACE homepage. http://www.caip.rutgers.edu/TASSL.

[31] E. Aprà, T. L. Windus, T. P. Straatsma, E. J. Bylaska, W. de Jong, S. Hirata, M. Valiev, M. Hackler, L. Pollack, K. Kowalski, R. Harrison, M. Dupuis, D. M. A. Smith, J. Nieplocha, V. Tipparaju, M. Krishnan, A. A. Auer, E. Brown, G. Cisneros, G. Fann, H. Früchtl, J. Garza, K. Hirao, R. Kendall, J. Nichols, K. Tsemekham, K. Wolinski, J. Anchell, D. Bernholdt, P. Borowski, T. Clark, D. Clerc, H. Dachsel, M. Deegan, K. Dyall, D. Elwood, E. Glendening, M. Gutowski, A. Hess, J. Jaffee, B. Johnson, J. Ju, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield, X. Long, B. Meng, T. Nakajima, S. Niu, M. Rosing, G. Sandrone, M. Stave, H. Taylor, G. Thomas, J. van Lenthe, A. Wong, and Z. Zhang. *NWChem, A Computational Chemistry Package for Parallel Computers, Version 4.7*. Pacific Northwest National Laboratory, Richland, Washington 99352–0999, USA, 2005.

[32] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global Arrays: A non-uniform-memory-access programming model for high-performance computers. *J. Supercomputing*, 10(2):169, 1996.

[33] M. Krishnan, Y. Alexeev, T. L. Windus, and J. Nieplocha. Multilevel parallelism in computational chemistry using Common Component Architecture and Global Arrays. In *Proceedings of Supercomputing*, 2005.

[34] Matthew Farthing, David Sassen, Jan F. Prins, and Cass T. Miller. A problem solving environment for subsurface flow and transport phenomena. In *International Conference on Computational Methods in Water Resources XV*. Elsevier, 2004.

[35] S. Zhou, W. Kuang, W. Jian, P. Gary, J. Palencia, and G. Gardner. High-speed network and grid computing for high-end computation. *Intl. J. of Concurrency and Comput.: Practice and Experience*, 2006. in press.

[36] W. Kuang and J. Bloxham. Numerical modeling of magnetohydrodyanmic convection in a rapidly rotating spherical shell: weak and strong field dynamo actions. *J. Comp. Phys*, 153:51–81, 1999.

[37] W. Kuang and B. F. Chao. Geodynamo modeling and core-mantle interaction. In Dehandt, Creager, Karato, Zatman, and AGU, editors, *Earth's Core: Dynamics, Structure, Rotation, Geodynamics*, 31, pages 193–212. Amer. Geophys. Union, 2003.

[38] Sameer Shende and Allen D. Malony. The TAU Parallel Performance System. *Intl. Journal of High-Performance Computing Applications*, ACTS Collection Special Issue, Summer 2006, 2006.

[39] Sparskit: A basic tool-kit for sparse matrix computations, version 2. `http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html`.

[40] SciDAC Terascale Optimal PDE Simulation (TOPS) center. `http://tops-scidac.org`.

[41] SciDAC Terascale Optimal PDE Simulation (TOPS) center Solver Component. `http://www-unix.mcs.anl.gov/scidac-tops/solver-components/tops.html`.

[42] ITSOL: Sparse iterative solvers package. `http://www-users.cs.umn.edu/~saad/software/ITSOL/`.

[43] The Massively Parallel Quantum Chemistry program. `http://www.mpqc.org/`.

[44] S. Benson, L. C. McInnes, J. Moré, and J. Sarich. TAO users manual. Technical Report ANL/MCS-TM-242 - Revision 1.8, Argonne National Laboratory, 2005. `http://www.mcs.anl.gov/tao/`.

[45] S. Balay, K. Buschelman, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, Barry F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.3.1, Argonne National Laboratory, 2006. `http://www.mcs.anl.gov/petsc`.

[46] David E. Bernholdt, Benjamin A. Allan, Robert Armstrong, Felipe Bertrand, Kenneth Chiu, Tamara L. Dahlgren, Kostadin Damevski, Wael R. Elwasif, Thomas G. W. Epperly, Madhusudhan Govindaraju, Daniel S. Katz, James A. Kohl, Manoj Krishnan, Gary Kumfert, J. Walter Larson, Sophia Lefantzi, Michael J. Lewis, Allen D. Malony, Lois C. Mclnnes, Jarek Nieplocha, Boyana Norris, Steven G. Parker, Jaideep Ray, Sameer Shende, Theresa L. Windus, and Shujia Zhou. A component architecture for high-performance scientific computing. *International Journal of High Performance Computing Applications*, 20(2):163–202, 2006.

[47] Joseph P. Kenny, Steven J. Benson, Yuri Alexeev, Jason Sarich, Curtis L. Janssen, Lois Curfman Mcinnes, Manojkumar Krishnan, Jarek Nieplocha, Elizabeth Jurrus, Carl Fahlstrom, and Theresa L. Windus. Component-based integration of chemistry and optimization software. *J. of Computational Chemistry*, 24(14):1717–1725, 15 November 2004.

[48] Douglas C. Schmidt and Fred Kuhns. An overview of the real-time CORBA specification. *Computer*, 33(6):56–63, June 2000.

[49] M. Díaz, D. Garrido, S. Romero, B. Rubio, E. Soler, and J.M. Troya. Nuclear power plant simulators: A component-based approach. In *Proceedings of Applied Simulation and Modelling - 2005*, Benalmádena, Spain, 15 – 17 June 2005. IASTED.

[50] E. F. Jaeger, L. A. Berry, and D. B. Batchelor. Second-order radio frequency kinetic theory with applications to flow drive and heating in tokamak plasmas. *Phys. Plasmas*, 7:641–656, 2000.

[51] A. J. Kemball, R. M. Crutcher, and R. Hasan. A component-based framework for radio-astronomical imaging software systems. In submission.

[52] Terascale simulation tools and technologies (TSTT) center. `http://www.tstt-scidac.org/`.

[53] Boyana Norris, Satish Balay, Steve Benson, Lori Freitag, Paul Hovland, Lois McInnes, and Barry Smith. Parallel components for PDEs and optimization: Some issues and experiences. *Parallel Computing*, 28:1811–1831, 2002.

[54] Scott Kohn, Gary Kumfert, Jeff Painter, and Cal Ribbens. Divorcing language dependencies from a scientific software library. In *Proceedings of the 10th SIAM Conference on Parallel Processing*, Portsmouth, VA, March 2001.

[55] Lois Curfman McInnes, Benjamin A. Allan, Robert Armstrong, Steven J. Benson, David E. Bernholdt, Tamara L. Dahlgren, Lori Freitag Diachin, Manojkumar Krishnan, James A. Kohl, J. Walter Larson, Sophia Lefantzi, Jarek Nieplocha, Boyana Norris, Steven G. Parker, Jaideep Ray, and Shujia Zhou. Parallel PDE-based simulations using the common component architecture. In Are Magnus Bruaset, Petter Bjørstad, and Aslak Tveito, editors, *Numerical Solution of PDEs on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering (LNCSE)*, pages 327–384. Springer-Verlag, 2006. Also available as Argonne National Laboratory technical report ANL/MCS-P1179-0704.

[56] Fang Peng, Meng-Shiou Wu, Masha Sosonkina, Ricky A. Kendall, Michael W. Schmidt, and Mark S Gordon. Coupling GAMESS via standard interfaces. In *Proc. of HPC-GECO/Compframe 2006 Joint Workshop*, Paris, France, June 2006.

[57] Computational models for studying electorn effects in heavy ion fusion accelerators. `http://www.txcorp.com/`

projects/\#projects_HIFA.
[58] P. H. Stoltz, S. A. Veitzer, R. H. Cohen, A. W. Molvick, and J. L. Vay. Energy loss, range, and electron yeild comparisons of the CRANGE ion-material interaction code. *Nuclear Instruments and methods in Physics Research*, 544:502–505, 2005.
[59] Paul Dubois. Pyfort: The Python–Fortran connection tool. `http://pyfortran.sourceforge.net/`, August 2002.
[60] SWIG: Simplified Wrapper and Interface Generator. `http://www.swig.org`.
[61] Pearu Peterson. F2PY: Fortran to python interface generator. `http://cens.ioc.ee/projects/f2py2e`, January 2005.
[62] Petascale simulation initiative website. `http://www.llnl.gov/CASC/psi`.
[63] Gary Kumfert, James Leek, and Thomas Epperly. Babel remote method invocation. In Submission.
[64] Dan Quinlan, Qing Yi, Gary Kumfert, Thomas Epperly, Tamara Dahlgren, Markus Schordan, and Brian White. Toward the automated generation of components from existing source code. In *Second Workshop on Productivity in High-End Computing*, San Francisco, CA, February 2005.
[65] C. E. Rasmussen, M. J. Sottile, S. S. Shende, and A. D. Malony. Bridging the language gap in scientific computing: the Chasm approach. *Concurrency and Computation: Practice and Experience*, 18(2):151–162, 2006.
[66] Chasm: Language Interoperability Tools. `http://chasm-interop.sourceforge.net/`.
[67] Dan Quinlan. Rose compiler infrastructure website. `http://www.llnl.gov/CASC/rose`, 2006.
[68] Eclipse IDE website. `http://www.eclipse.org`.
[69] Welf Löwe and Thomas Panas. Rapid construction of software comprehension tools. *Intl. Journal of Software Engineering and Knowledge Engineering: Special Issue on Maturing the Practice of Software Artefacts Comprehension*, 12(54), 2005.
[70] Thomas Panas. *A framework for reverse engineering*. PhD thesis, Växjō University, Sweden, December 2005.
[71] Component object model. `http://www.microsoft.com/com/tech/com.asp`, 2003.
[72] Enterprise Java Beans. Enterprise Java Beans. `http://java.sun.com/products/javabeans`, 2003.
[73] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit, An Object-Oriented Approach to 3-D Graphics*. Prentice Hall PTR, 2nd edition, 2003.
[74] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *16th Intl. Conf. on Scientific and Statistical Database Management (SSDBM'04)*, Santorini Island, Greece, 2004.
[75] Kepler project. `http://kepler-project.org`.
[76] S.G. Parker and C.R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Supercomputing '95*. IEEE Press, 1995.
[77] S.G. Parker, D.M. Beazley, and C.R. Johnson. Computational steering software systems and strategies. *IEEE Computational Science and Engineering*, 4(4):50–59, 1997.
[78] Madhusudhan Govindaraju, Himanshu Bari, and Michael J. Lewis. Design of distributed component frameworks for computational grids. In *The International Conference on Communications in Computation*, pages 160–166, June 2004.
[79] Maciej Malawski, Dawid Kurzyniec, and Vaidy Sunderam. MOCCA - towards a distributed CCA framework for metacomputing. *ipdps*, 05:174a, 2005.
[80] Earth System Modeling Framework website. `http://www.esmf.ucar.edu`, 2006.
[81] N. Collins, G. Theurich, C. DeLuca, M. Suarez, A. Trayanov, V. Balaji, P. Li, W. Yang, C. Hill, and A. da Silva. Design and implementation of components in the earth system modeling framework. *Intl. J. High-Perf. Computing Appl.*, 19(3):341–350, Fall 2005.
[82] C. Hill, C. DeLuca, V. Balaji, M. Suarez, and A. da Silva. The architecture of the earth system modeling framework. *Computers in Science and Engineering*, 6(1):18–28, 2004.
[83] S. Zhou. Coupling climate models with earth system modeling framework and common component architecture. *Concurrency and Computation: Practice and Experience*, 18:203, 2006.
[84] J. Walter Larson, Boyana Norris, Everest T. Ong, David E. Bernholdt, John B. Drake, Wael R. Elwasif, Michael W. Ham, Craig E. Rasmussen, Gary Kumfert, Daniel S. Katz, Shujia Zhou, Cecelia DeLuca, and Nancy S. Collins. Components, the common component architecture, and the climate/weather/ocean community. In *84th American Meteorological Society Annual Meeting*, Seattle, Washington, 11–15 January 2004. American Meteorological Society.
[85] Terascale Supernova Initiative website. `http://www.phy.ornl.gov/tsi/`, 2006.
[86] F. Douglas Swesty and Eric S. Myra. Multigroup models of the convective epoch in core collapse supernovae. In Anthony Mezzacappa, editor, *SciDAC 2005, Scientific Discovery through Advanced Computing, 26–30 June 2005, San Francisco, CA, USA*, volume 16 of *Journal of Physics: Conference Series*, pages 380–389. Institute of Physics, 2005.
[87] Lois Curfman McInnes, Jaideep Ray, Rob Armstrong, Tamara L. Dahlgren, Allen Malony, Boyana Norris, Sameer Shende, Joseph P. Kenny, and Johan Steensland. Computational quality of service for scientific CCA applications: Composition, substitution, and reconfiguration. Technical Report ANL/MCS-P1326-0206, Argonne National

Laboratory, Feb 2006. Available via ftp://info.mcs.anl.gov/pub/tech_reports/reports/P1326.pdf.

[88] B. Norris, J. Ray, R. Armstrong, L. C. McInnes, D. E. Bernholdt, W. R. Elwasif, A. D. Malony, and S. Shende. Computational quality of service for scientific components. In *Proc. of International Symposium on Component-Based Software Engineering (CBSE7), Edinburgh, Scotland*, 2004.

[89] P. Hovland, K. Keahey, L. C. McInnes, B. Norris, L. F. Diachin, and P. Raghavan. A quality of service approach for high-performance numerical components. In *Proceedings of Workshop on QoS in Component-Based Software Engineering, Software Technologies Conference, Toulouse, France*, June 2003. (also available as Argonne preprint ANL/MCS-P1028-0203).

[90] J. Ray, N. Trebon, S. Shende, R. C. Armstrong, and A. Malony. Performance measurement and modeling of component applications in a high performance computing environment : A case study. In *Proceedings of the 18^{th} International Parallel and Distributed Processing Symposium*, Los Alamitos, California, USA, April 2004. IEEE Computer Society. also Sandia National Laboratories Technical Report SAND2003-8631, November 2003.

[91] A. Malony, S. Shende, N. Trebon, J. Ray, R. Armstrong, C. Rasmussen, and M. Sottile. Performance Technology for Parallel and Distributed Component Software. *Concurrency and Computation: Practice and Experience*, 17:117–141, Feb–Apr 2005.

[92] N. Trebon, A. Morris, J. Ray, S. Shende, and A. Malony. Performance modeling of component assemblies with TAU. Presented at Compframe 2005 workshop, Atlanta, June, 2005.

[93] Performance Evaluation Research Center (PERC). http://perc.nersc.gov/.

[94] J. Amundson, P. Spentzouris, J. Qiang, and R. Ryne. Synergia: An accelerator modeling tool with 3-D space charge. *Journal of Computational Physics*, 211:229–248, January 2006.

[95] Douglas Dechow (PI). A beam dynamics application based on the common component architecture. Phase-I SBIR Project, June 2006.

[96] Tamara L. Dahlgren and Premkumar T. Devanbu. Improving scientific software component quality through assertions. In *Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications*, pages 73–77, St. Louis, Missouri, May 2005. Also available as Lawrence Livermore National Laboratory Technical Report UCRL-CONF-211000, Livermore, CA, 2005.

[97] Tamara L. Dahlgren. Adaptive enforcement of component interface assertions. Technical report, Lawrence Livermore National Laboratory, Livermore, California, 2006. in progress.