Center for Component Technology for Terascale Simulation Software

SciDAC Review Report (March 2003)

Principal Investigator:

Rob Armstrong (SNL)

Co-Investigators:

David Bernholdt (ORNL)
Lori Freitag Diachin (SNL)
Dennis Gannon (Indiana University - IU)
James Kohl (ORNL)
Gary Kumfert (LLNL)
Lois Curfman McInnes (ANL)
Jarek Nieplocha (PNNL)
Steven Parker (University of Utah - UU)
Craig Rasmussen (LANL)

Further Information:

Rob Armstrong
Sandia National Laboratory
7011 East Avenue
Livermore, CA 94551-0969
phone: 925-294-2470
fax: 925-294-1225
email: rob@sandia.gov

http://www.cca-forum.org/ccttss

Contents

1. Introduction

The Center for Component Technology for Terascale Simulation Software (CCTTSS) is dedicated to accelerating science by bringing a "plug-and-play" style of programming to high-performance computing. Through our programming model called the Common Component Architecture (CCA) [1, 2], we can dramatically reduce the time and effort required to compose independently created software libraries into new terascale applications. We have two major early-adopters of our technology in the application areas of combustion and quantum chemistry. A third application group, the Community Climate System Model, has CCA-enabled prototypes and are planning to adopt the CCA. By design, the customer sacrifices almost no performance when using CCA components. Moreover, components that are written in various languages, such as Fortran, C++, and Python, all work together transparently.

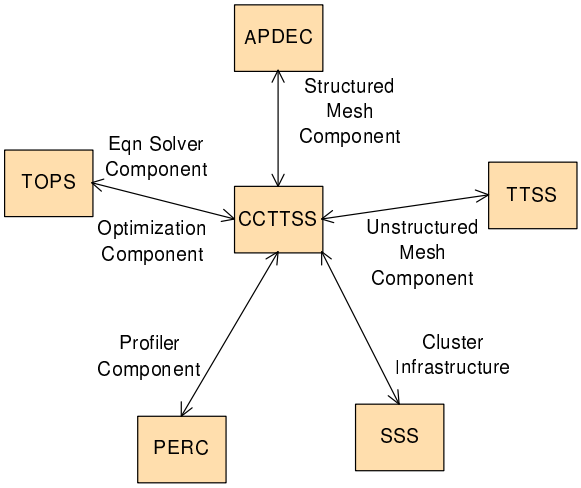

Figure 1.1: Diagram showing the relationship of the CCTTSS to other SciDAC ISICs and key interactions.

The CCTTSS center was formed around the grass roots Common Component Architecture (CCA) Forum, already in existence for three years prior to the start of the SciDAC initiative. The purpose of the CCA and the tools provided by the CCTTSS is to bring the software component paradigm to high-performance computing, thereby reducing barriers to scientific software reuse and enabling scientists to be more productive when constructing their codes. This mode of programming is common in industry, but was heretofore unknown in high-performance scientific computing. The CCA effort distinguishes itself from those in the commercial sector because we address the challenges associated with maintaining high performance, work with a broad spectrum of scientific programming languages and computer architectures, and aim to ensure that the DOE investment in legacy codes is preserved.

1.1 CCA Usage in SciDAC Applications

The vision behind the CCA specification is to ensure that compositions of compliant components will yield truly "plug-and-play" applications. If a scientific programmer follows the "rules" that are laid out in the CCA specification, or uses the tools supplied by the CCTTSS, then the resulting component is guaranteed to work with any other that is also CCA compliant. This paradigm enables a fundamental shift in the software development process. Rather than a handful of 'hero' programmers creating a monolithic executable, many developers and domain experts can collaboratively contribute components, which can then be assembled and reused in production scientific simulations.

Nothing illustrates this vision better than the software "facility" being developed by the SciDAC application center, A Computational Facility for Reacting Flow Science (CFRFS) [3], led by H. Najm. Just as physical laboratories supply scientific facilities to visiting researchers, so this facility of CCA-compliant software components provides the building blocks of combustion simulations. As shown by the wiring diagram in Figure 1.2, investigators can compose applications from these tools and add more components to prove a scientific point or otherwise increase the value of the facility.

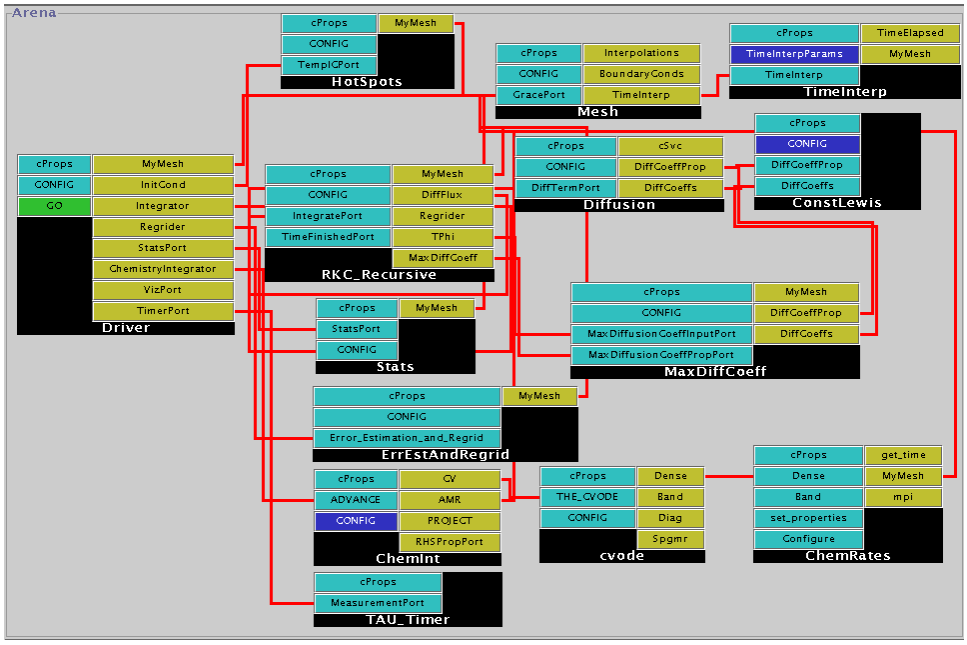

Figure 1.2: Wiring diagram of a component-based combustion application (CFRFS), which involves components for chemical and physical models as well as parallel structured adaptive mesh refinement, time integration, and performance monitoring.

In addition to this work, CCTTSS researchers are collaborating closely with application scientists to create massively parallel simulation codes in the areas of quantum chemistry and climate modeling.

1.2 The CCA Model and Terminology

A CCA *component* is a piece of working software (a few subroutines, a library, even a whole application), which is packaged as a unit that adheres to the CCA specification. The current version of the CCA specification is available via [4]. We often talk about programming interfaces in a generic sense, but the CCA also uses the term *interface* to refer to domain-specific standards that components can implement. A key concept in CCA is the notion of *ports*, which are specific interfaces that are exchanged when components are connected. These connections are always made between a *provides port* and a *uses port*. A CCA *framework* is the hosting runtime environment where components are created and managed.

How components behave at runtime is different from the usual behavior of structured or object-oriented programming. When CCA components are created, they know only how to communicate with the framework. They post what ports they provide, what ports they use, and wait. If the components are connected by some third party (typically the user, through a GUI or script), and commands are invoked on them through their *provides* ports, the components acquire whatever *uses* ports they previously posted (if any), service the command, and return. Hence, each component is written to depend only on explicit standard interfaces (ports), never naming or knowing the underlying software on which they are relying, only the standard that one uses and the other provides.

1.3 CCA Technology Development

To enable CCA-compliant applications to function properly, the CCTTSS is addressing a number of challenging research questions. In particular, we are pursuing ongoing work in framework and infrastructure development (including language interoperability), scientific component design, and parallel data redistribution.

Frameworks and Infrastructure. The CCTTSS maintains three frameworks, each of which explores different areas of scientific computing. Ccaffeine [5] concentrates on the Single Program Multiple Data (SPMD) paradigm and massive parallelism, while XCAT [6] supports distributed computing, emphasizing Grid support. SCIRun [7] supports both parallelism models to some extent but also handles multi-threaded cases. Because application scientists use a variety of programming languages, the CCA has been designed to be language neutral. Our Scientific Interface Definition Language (SIDL) [8] and associated compiler, Babel [9], allow components to be written and used from any of C, C++, Python, and Fortran with minimal performance costs. Chasm [10] is a set of automatic language transformation tools that ease the burden of developing components from existing code.

Scientific Components. As a part of its mission, the CCTTSS has developed production components that are used in scientific applications as well as prototype components that aid in teaching CCA concepts. This work has been recognized as one of the "Top Ten Science Achievements in 2002" by the DOE Office of Science [11]. These freely available components include various service capabilities, tools for mesh management, discretization, linear algebra, integration, optimization, parallel data description and redistribution, visualization, and performance evaluation. To define common interfaces for mesh-based scientific data management, the CCTTSS is collaborating with domain experts in two SciDAC Integrated Software Infrastructure Centers (ISICs), namely the the Terascale Simulation Tools and Technologies (TSTT) ISIC [12], led by J. Glimm, and the Algorithmic and Software Framework for Applied PDEs (APDEC) ISIC [13], led by P. Colella. These data-centric components will define the parallel layout of data across processors and form the *lingua franca* for other components that manipulate the data. We are also working in conjunction with the Terascale Optimal PDE Simulations (TOPS) ISIC [14], led by D. Keyes, to define component interfaces for linear, nonlinear, and optimization solvers. Figure 1.1 illustrates such interactions among SciDAC ISICs from the CCTTSS perspective.

"MxN" Parallel Data Redistribution. A common barrier to scientific software reuse is a disparate underlying model for how data is distributed in memory by different codes. MxN tools provide the basic functions for exchanging parallel data, including describing and registering the data, creating connections with parallel "communication schedules," and then actually transferring parallel data elements among components. This basic yet challenging work will be the foundation for a suite of model coupling capabilities, including spatial and temporal interpolation and unit conversions. Such manipulation of parallel data also enables Parallel Remote Method Invocation (PRMI), wherein the input and output arguments to method calls can themselves be distributed data objects.

CCA Outreach Activities. Much of the CCA's success thus far is due to the outreach done by the CCTTSS in the SciDAC and larger computational science community. During the past year, we conducted eight tutorials in four time zones and have provided extensive documentation on our web site, `http://cca-forum.org`.

1.4 Benefits from SciDAC

The CCA reaps a two-fold benefit from the SciDAC initiative: accelerating both the rate of CCA technology development and the insertion of this technology into massively parallel scientific applications.

Internally, SciDAC has catalyzed a loose confederation of grass-roots interests into a coherent team. Before SciDAC, we had been sharing ideas. Now, we are sharing code, co-developing software, and closely coordinating efforts. Integration of our previously disparate efforts is a recurring theme throughout this review document.

Externally, SciDAC's emphasis on vertical integration — infrastructure impacting applications, and applications effectively leveraging investments in infrastructure — has created an ideal environment for the CCA. The CCA exists to achieve vertical software integration, though its members tend to express this as "software reuse." After one year of funding, CCA components and infrastructure are used in two major SciDAC applications, numerous CCA-compliant application components have been developed and deployed, and the underlying infrastructure is maturing and establishing itself in the SciDAC scientific community.

2. Progress, Application Impact, and Future Plans

2.1 Frameworks and Infrastructure

Coordinator: G. Kumfert (LLNL)

CCA frameworks are the foundation of CCA component applications. The frameworks effort of the CCTTSS proposal is arguably the most speculative of the facets of the project, simply because there is no prior art for component frameworks in high-performance scientific computing.

Given the speculative nature of this effort, progress to date has been remarkably close to, and often exceeding, expectations. Of the three primary goals for a CCA component framework to satisfy [15] — support for the SPMD paradigm, distributed computing via the Grid, and commodity architectures (e.g., Enterprise Java Beans [16, 17], CORBA [18–21], Microsoft COM [22, 23], or Microsoft .NET) — we have completed the first, demonstrated the second, and are well-positioned for the third.

2.1.1 Framework Implementations

The CCTTSS supports three frameworks, each with its own specialties and purpose. Linkages among these frameworks provide a full complement of features to work under various application scenarios. Of the three, Ccaffeine [5] is most production oriented and best positioned for most terascale simulations, as it supports the Single Component Multiple Data (SCMD) paradigm, which is the component analog of the SPMD programming style. By comparison, XCAT [6] is the most experimental, emphasizing distributed computation across the Grid. SCIRun [7] combines aspects of the other two, supporting experimental models of handling distributed SPMD components.

We discuss each of the frameworks separately in Sections 2.1.1.a–2.1.1.c, and we then discuss current efforts for interoperability among the frameworks in Section 2.1.1.d.

2.1.1.a Ccaffeine - SCMD Framework

Investigators: B. Allan, R. Armstrong (SNL)

Progress to Date. Ccaffeine has seen a substantial amount of use, technical development, and usability improvements since the beginning of SciDAC. In addition to the improvements in services for scientific components listed in Appendix A.1, advances in the framework itself are noteworthy. Of particular importance is Ccaffeine's new and unique ability to handle multiple types of components, depending on which (if any) language interoperability tool is used. This capability sprang from the innovation that the CCA design pattern is separate from the specification [24].

Originally, Ccaffeine was a C++ only tool; following the original C++ CCA specification is now called *Classic*. During the last year, Ccaffeine also implemented support for *SIDL components* — allowing components written in any Babel-supported language. This is now the strictly CCA-compliant (but not exclusive) way to use Ccaffeine. The original Classic binding continues to support *Classic components* for backward compatibility. A third mode, supporting automatically code generated *Chasm components*, was added most recently. As Chasm and Babel continue to coordinate their efforts, however, the need for Chasm as a sepa-

rate binding from SIDL is expected to go away. Translation services have been implemented so that Classic components and SIDL components can interoperate.

Ccaffeine also tracks the CCA specification most closely as it continues to evolve. Recent additions of BuilderServices, TypeMaps, and AbstractFrameworks have all been implemented and enable Ccaffeine to present itself as a component to external frameworks. Discussion about how this feature was demonstrated at SC2002 follows in Section 2.1.1.d. The framework can be used serially or on a parallel machine using MPI [25] or PVM [26] in any of three modes: interactively with a GUI, interactively at a command line, and in batch mode using a script derived from the first two modes. The wiring demo in Figure 1.2 (page 3) is a screenshot of Ccaffeine's GUI in action. Additionally, Ccaffeine can be used entirely as a library to allow users to script their own framework actions in Python or C++.

Impact on Applications. Ccaffeine is currently in use by a number of applications, including combustion simulations within the Computational Facility for Chemically Reacting Flow (see Section 2.4.2.a) and a quantum chemistry application using prototype optimization components from the TOPS ISIC (see Section 2.4.2.b). Ccaffeine is the framework that is used for the CCA tutorial hands-on codes, and almost all users who are interested in the CCA for SPMD-like parallel computing usually start with this framework.

2.1.1.b XCAT - Grid Framework

Investigator: D. Gannon (IU)

Progress to Date. XCAT [6] is an implementation of the CCA specification that runs on distributed, heterogeneous Grid platforms. The emerging Grid middleware standard is based on Web Services [27], which is a port-based architecture for coupling clients to servers over the Internet. This standard is supported by many major industrial concerns including IBM, Microsoft, Oracle, SUN, Intel and others. This web services standard was also influenced by the CCA, and there is a natural mapping of CCA concepts into the web services model [6]. The Global Grid Forum is also extending the web services standard to support Grid applications based on the Open Grid Services Architecture (OGSA) [28–33].

The CCA component model addresses application construction by component composition. Many of the resulting applications run in SPMD mode on scalable parallel systems. However, by design, there is nothing in the CCA specification that limits applications to being only SPMD. This allows CCA applications to be built from components running at different locations on the network. It also allows arbitrary OGSA web services to be used as components in a CCA application, and it allows CCA applications to appear as first-class, remote services to desktop web-service standard compliant frameworks such as Microsoft's .NET.

Impact on Applications. In a distributed, multi-scale simulation of the semiconductor fabrication process, several existing parallel simulations were "wrapped" as XCAT CCA components [32]. These components were run on different remote hosts, and CCA ports were used to move data between the simulations. Additionally, XCAT has identified a new project for creating an infrastructure for combining severe storm modeling with on-line sources of weather data. We expect to increase emphasis on science applications as Grid standards evolve and XCAT matures.

2.1.1.c SCIRun - Concurrency Framework

Investigator: S. Parker (UU)

Progress to Date. SCIRun [7] is the most established framework in the CCTTSS; it is used by a handful of government agencies, seven external universities, and five corporations. The SciDAC element of this project has focused on core CCA standards, including how to implement the CCA standard in a multi-threaded context as well as interoperability vehicles with XCAT and Ccaffeine. The CORBA concurrency service served as a starting point, but several areas have been identified that need extension in support of scientific computing. Most recently, SCIRun has contributed a prototype system that implements parallel data redistribution and Parallel Remote Method Invocation (PRMI) (see Section 2.3 for more details).

Impact on Applications. The current SCIRun distribution predates the CCA, and is not generally interoperable with Ccaffeine and XCAT. SCIRun2, however, will be a CCA complaint framework. SCIRun2 is already under development, incorporating SCIRun, Uintah [34, 35], and multi-threaded CCA prototypes. SCIRun is currently used for fire simulations, combustion modeling, remote visualization, bioelectric modeling, atmospheric diffusion, electrical impedance tomography, chemical engineering, cardiac defibrillation design, cell biology, medical imaging, and geophysical science simulations. We are optimistic about the CCA's impact on applications with the roll-out of SCIRun2.

2.1.1.d Frameworks Interoperability

The CCA is generic enough to support SPMD, distributed, and multi-threaded applications, as demonstrated by the CCTTSS' portfolio of frameworks. This provides considerable flexibility to users, and many challenges to framework creators, particularly because we want the frameworks themselves to be interoperable.

At SC2002 we demonstrated a proof of concept by connecting Ccaffeine and XCAT for a combustion application. Ccaffeine was used to compose and run parallel high-performance combustion simulations, and XCAT provided the connectivity for the simulation to be accessed as a Grid resource. Plans for continued progress in framework interoperability are discussed in Section 2.1.4

2.1.2 Language Interoperability Tools

There are two main language interoperability efforts in the CCTTSS that fill two very different niches. Babel [8] is an IDL-based tool that supports C, C++, Python, Java, Fortran77, and Fortran90. Chasm [10] is an automatic parsing technology that connects C++ and Fortran90.

2.1.2.a SIDL/Babel

Investigators: T. Dahlgren, T. Epperly, G. Kumfert (LLNL)

Progress to Date. During the past eighteen months, Babel has been adopted as the CCA's standard mechanism for language interoperability, thus making it an integral part of the CCA. The increased attention within the community has resulted in a significant increase in the need for fine-tuning and hardening Babel in addition to addressing research needs.

Our primary technical accomplishments over this period are (1) the addition and hardening of client-side Java, (2) the addition and hardening of client- and server-side Python, (3) the addition of Phase I client- and server-side Fortran90 support, (4) significant improvements in support of multi-dimensional arrays, (5) support for reentrant and unversioned packages, (6) the ability to generate SIDL Extensible Markup

Language (XML) encoded interface specifications, and (7) the addition of language-neutral support for overloaded method names.

Impact on Applications. The first impact Babel had to make under the SciDAC initiative was to introduce Babel into the CCA frameworks. Although the Livermore team participated in the CCA Forum from its inception, the Babel tool was not being used in framework development when SciDAC started. Thus, at the Fall 2001 CCA meeting, we unveiled Decaf [36, 37] as the first Babel-enabled CCA compliant framework. Ccaffeine added Babel support shortly thereafter; reusing most of the Decaf implementation in the process. XCAT and SCIRun have more complex distributed computing features that Babel must integrate first, before these two frameworks can logically adopt Babel directly, though some prototyping with SCIRun is in progress.

Next, Babel had to impact component writers. This happened very quickly after Decaf had been released, and Ccaffeine announced intent to support Babel. At SC2001, Babel was not used in any of the CCA demos. By SC2002, practically half of the CCA demos were running Babelized components in Ccaffeine. All of the scientific components developed at Argonne that had initially been written in the so-called "classic" style have now been rewritten as SIDL components. In addition, collaborators within the Terascale Optimal PDE Simulations (TOPS) SciDAC Center (PI: D. Keyes) have adopted SIDL for defining new interfaces for numerical software.

Fortran90: A Mid-Course Correction. Support for Fortran90 has exceeded all initial expectations of scope, importance, impact, and complexity. The initial proposal casually lists Fortran90 support as a second year deliverable [15, pg 23]. At the time, the thought was that this would be of moderate importance, and that a minimal connectivity solution would be sufficient. As the SciDAC community started forming, we quickly began to appreciate just how many SciDAC applications are written in Fortran90, and how many SciDAC math and CS libraries are not. Furthermore, starting with an open email to the CCA from Randall Bramley [38], the CCTTSS became increasingly aware that we had underestimated the number of Fortran90 features that were commonly used in scientific programs.

Corrective action is already well underway. We maintain constant interaction with the Chasm team and other interested parties through a mailing list dedicated to CCA/Fortran90 issues. We have rolled out our initial vision for Fortran90 support, but now refer to it merely as "Phase I." We have already received valuable feedback from the community, and we have implemented most of it as Phase II in our upcoming release. Contrary to the original proposal, we see Fortran90 development continuing for the foreseeable future in this iterative and communal refinement process.

Open Babel: A Mid-Course Correction. As originally conceived, Babel was to be an easy-to-use tool with an input grammar (SIDL) and a collection of backend code generators. As Babel gained acceptance, it also accumulated high expectations, and even SciDAC collaborators interested in doing research on SIDL/Babel itself — including applying SIDL/Babel to problems other than (or only loosely related to) language interoperability.

Open Babel [39] is an effort to engage a community of developers to contribute to Babel's continued success and increased diversity. There have been two meetings so far dedicated exclusively to Babel development issues. We have defined three levels of Babel development: Core, Extensions, and Dialects, which clearly define levels of interoperability guaranteed by Babel as well as flexibility for independent research on the part of the developer community. Babel Extensions is the target level that best allows new features to be "plugged-in" to Babel while still maintaining Babel's interoperability guarantee. This effort is still very much in the formative stages, focusing on community, ideas, and consensus building. Actual software contributions are just appearing on the horizon.

2.1.2.b Chasm

Investigators: M. Sottile, C. Rasmussen (LANL)

Progress to Date. Unlike many other aspects of the CCA, Chasm did not exist before SciDAC funding initiated. Chasm is a compiler-based research project that parses existing C++ and Fortran90 source files, automatically generating bindings between C++ functions and Fortran90 procedures. It also creates CCA components out of existing Fortran files. As mentioned in Section 2.1.1.a, support for Chasm has already been added in Ccaffeine. Recently, Chasm has added the capability to generate SIDL files from existing source code. These SIDL files can then be used by Babel to generate language bridging files.

The primary focus of Chasm within the CCA is to provide ease-of-use tools to help application developers transition to component-based applications. The first release of Chasm provided a C library that allows native-language arrays to be passed as function parameters between C++ and Fortran90. For example, a C++ programmer using this library can create a C++ array class and then pass this class to Fortran, where it is interpreted as a Fortran90 assumed-shape array.

Impact on Applications. The Chasm team works with Fortran application developers to define CCA Fortran bindings that appear natural to Fortran programmers. This work has already affected and continues to influence Babel's own Fortran90 bindings. Babel also plans to reuse Chasm's infrastructure for portable Fortran90 array descriptor manipulation.

2.1.3 Component Repository and Software Deployment

Investigators: T. Epperly (LLNL), B. Allan (SNL), B. Norris (ANL)

Progress to Date. The CCA and the CCTTSS emphasize the runtime interoperability of software, which is an important, but not exclusive source of software incompatibilities. Another source of incompatibilities is how software is packaged and built. This effort focuses on defining standards for describing, documenting, and distributing component software in source-code form to help address this second source of difficulties. Three major thrusts in this arena cover immediate needs as well as longer-term goals.

The short-term effort has focused on using an existing tool called Redhat Package Manager (RPM) to generate RPMs for the Ccaffeine framework, Babel, and various components demonstrated at SC2002 (see Section 2.2.2). We plan to make these publicly available on the CCA website [40] by early March, 2003. It is important to stress, however, that RPMs are a stopgap measure and not a long term strategy for all future distributions of component technology.

Our long-term efforts in this area have focused on development of an XML description of a component and updating our prototype component repository, Alexandria [41].

Impact on Applications. Given the early stage of the CCA, activity is focusing on framework and component development. As emphasis shifts to managing a large collection of existing components, we expect this area to have a substantial impact. In the short term, we expect the RPMs to provide the most accessible means for beginners to explore CCA component technology. Alexandria and XML encodings will be the focus for production use of CCA technology.

2.1.4 Future Plans

Future R&D efforts for Frameworks are summarized on Table 3.1 (page 29). The primary focus of all framework activity is to complete interoperability efforts that are already underway.

1. We will specify a remote interoperability network protocol that will serve as the foundation for remote method invocation between two components on dissimilar frameworks and machines. This protocol will be compatible with standard web services protocols, but have extensions for high performance.
2. We will deliver tools to transform SIDL interface specifications into running software that connects component ports from different frameworks using the aforementioned protocols.
3. We will work with the MxN group to ensure that this set of protocols and tools supports the MxN parallel coupling models that they are spearheading.

Babel and Chasm. Chasm's sourcecode to SIDL capabilities promise to lower the barrier for componentization of legacy codes. Chasm's Fortran90 infrastructure will be reused inside Babel to bolster its own. At this point, a Chasm user must hand edit Babel-generated implementation files to connect to existing library code. In time, this last hand-editing step will not be required.

Ccaffeine. In the near future Ccaffeine will be upgraded to include rendezvous of multiple instances of itself to form multi-SPMD programs. This capability is very much in demand in the climate community, which is the CCA's next big applications effort.

2.2 Scientific Components

Coordinator: Lois Curfman McInnes (ANL)
Leader of Scientific Data Components Working Group: Lori Freitag Diachin (SNL, formerly ANL)
Investigators: See Appendix A for the developers of various components.

Before we can begin to realize our vision of interoperable computational science components, we must address many difficult research issues. We employ a four-pronged approach to scientific component development, namely (1) defining domain-specific interface specifications through collaborations with other SciDAC centers; (2) developing a suite of parallel scientific components; (3) collaborating with applications scientists to evaluate and extend these interfaces and components; and (4) exploring research issues that are unique to the DOE computational science environment, including quality of service issues related to robust, efficient, and scalable performance. We began the SciDAC initiative with a strong foundation for this work in the form of rich parallel tools that already used abstractions in their design. This section discusses progress during the past eighteen months.

2.2.1 Common Interface Development

The development of common interfaces that a large number of tools support is critical to the notion of "plug-and-play" scientific computing for which the CCTTSS is striving. Common interfaces are especially important for scientific data components, as the interfaces for numerical tools and application components often include a certain degree of specificity of the data structures used as input and output arguments. Over the past year, we have worked with domain experts in the TSTT [12] and APDEC [13] ISICs to develop common interfaces for two broad areas of scientific data components, namely a descriptor for dense arrays distributed across the processors of a parallel computer (see Section 2.2.1.a) and interfaces for structured, unstructured, and adaptive mesh access (see Section 2.2.1.b). These data-centric components will define the parallel layout of data across processors and form the *lingua franca* for other components that manipulate the data. More recently, we have also begun to work in conjunction with the TOPS ISIC [14] to define compo-

nent interfaces for linear, nonlinear, and optimization solvers. In addition, we engage the high-performance scientific community at large to participate in similar dialogues in their particular areas of expertise.

2.2.1.a Distributed Arrays

In scientific simulations on parallel computers, arrays that are distributed across the memory of the machine are a widely used abstraction. At the same time, different codes often have their own distinct way of implementing that abstraction. The Distributed Array Descriptor (DAD) interface was developed to facilitate interoperability among components utilizing the distributed array abstraction by providing a "common currency" through which components can describe, exchange, and access distributed arrays in a way that is independent of the underlying implementation.

The interface specification was developed by a sub-group of the CCA Forum's Scientific Data Components Working Group over the course of approximately a year. The design took into consideration many distributed array implementations, including High Performance Fortran (HPF) [42, 43], PETSc [44, 45], LAPACK [46], KeLP [47], P++ [48], the Global Array Toolkit [49–51], and others. The resulting design drew particularly on the HPF distributed array model, which was determined to provide a superset of capabilities provided by most of the other distributed array packages.

Based on the group's design, David Bernholdt (ORNL) produced a reference implementation of the interface, the Distributed Array Descriptor Factory (DADF) component, which was first used in demonstration applications shown at SC2001. The DAD interface and the DADF reference implementation have been used both to allow interoperability of components based on existing software with different distributed array implementations, and as the basis for the development of new capabilities. The "MxN" parallel data redistribution effort (see Section 2.3) has developed interfaces for that problem in which distributed arrays described using the DAD are the input and output. Similarly, the Global Array Toolkit has been adapted to use the DAD as its primary interface – not simply to *describe* existing distributed arrays, but to actually use the DAD representation to *create* new distributed arrays (see Section 2.2.2).

The original DAD interface was designed before Babel was mature enough for routine use, so Wael Elwasif (ORNL) and David Bernholdt (ORNL) are in the process of adapting the original specification to an appropriate SIDL variant, which will become the standard. The DADF reference implementation is likewise being updated.

2.2.1.b Computational Meshes

In conjunction with the TSTT and APDEC ISICs, we have worked to create common interfaces that are appropriate for the representation of structured, unstructured, and adaptive meshes. As with the Distributed Arrays effort, our focus is the definition of accessor interfaces that provide the functionality needed by application programmers independent of the representation in the underlying implementation. The challenges inherent in this type of effort include balancing the performance of the interface with the flexibility needed to support a wide variety of mesh types and the desire to keep the interface minimal so that it is simple to implement and adopt.

In our work with the TSTT interface definition group, we have focused primarily on accessing geometry and topology information from both structured and unstructured static meshes. To provide the flexibility needed to support a wide array of mesh types, access to information is provided through a number of different mechanisms. For example, care was taken to ensure that access to coordinate and adjacency information in the mesh was provided on both an entity-by-entity basis by using iterators of opaque objects and for the entire mesh at once through arrays of doubles. All implementations are required to support both modes of access, although it is recognized that only one style is typically native to the underlying software. Implementations of this specification are underway at LLNL (for Overture [52]), PNNL (for NWGrid [53]), RPI (for

AOMD [54]), SNL (for MDB/CUBIT [55]), and ANL (for a simple triangular mesh). A TSTT-compliant mesh component was first used and demonstrated for SC2001 for solving simple PDE-based applications and continues to be used and distributed as part of the CCA tutorial. Additional efforts that use this interface are ongoing at SUNY Stony Brook as part of the Frontier [56] merge with various TSTT mesh generation technologies and as part of the MESQUITE mesh quality improvement toolkit [57].

In addition, in April of 2002, the CCA hosted a joint meeting with the APDEC and TSTT Centers to define a common interface that supports the particular needs of Structured Adaptive Mesh Refinement (SAMR) data structures. The primary goal of this interface is to allow the multiple tools that implement this technique to exchange data and use each other's solvers and pre- and post-processing tools. For example, in the combustion application described previously, the use of Chombo [58] solvers (developed at LBNL) is critical to the long term success of the project, and a common interface that allows access to them needs to be developed. At this meeting a preliminary interface was adopted, and a reference implementation is currently underway at LBNL.

2.2.2 Prototype Components

As a part of its mission, the CCTTSS has developed (internally and through collaboration) high-performance production components that are used in scientific applications as well as prototype components that aid in teaching CCA concepts. A common theme in this work is combining application-specific components (which are further discussed in Section 2.4) with more general-purpose ones that can be reused across a range of applications. These freely available components include various service capabilities, tools for mesh management, discretization, linear algebra, integration, optimization, parallel data description and redistribution, visualization, graphical building, and performance evaluation. We have also developed a variety of component-based scientific applications that demonstrate component reusability and composability, including several that employ domain-specific interfaces that are being defined by CCA working subgroups for scientific data and parallel data redistribution. Several of these applications solve PDE-based models using either adaptive structured meshes or unstructured meshes, while others solve unconstrained minimization models that arise in computational chemistry.

This work has been recognized as one of the "Top Ten Science Achievements in 2002" by the DOE Office of Science [11]. In addition, these components and applications, which we demonstrated at the SC2001 and SC2002 conferences, serve as part of CCA tutorial material and provide a starting point for interfaces with several scientific applications. All of these sample applications employ the Ccaffeine [5] framework, and two or more applications re-use in different contexts each of the components highlighted in Appendix A. These components leverage and extend parallel software tools developed at different institutions, including CUMULVS [59–62], Global Arrays [49, 50], GrACE [63], CVODES [64], MPICH [65], PETSc [44, 45], PVM [26], and TAO [66, 67].

To facilitate the exploration of CCA technology by potential new users while requiring a minimum of installation effort, we are releasing Linux binary and source RPMs for these components and supporting infrastructure (see Section 2.1.3). We provide highlights of some of these components in Appendix A; see [40] for the complete component suite, including source code and documentation via the RPMs.

2.2.3 Impact on Applications

2.2.3.a Numerical Components in Molecular Geometry Optimization

An important goal of our project is to assure that scientific components are interoperable, able to deliver high performance, and useful to real applications. As discussed in Appendix A, the `GlobalArray` and `TaoSolver` components offer complementary capabilities to their applications: the `GlobalArray` component (PNNL), which is based on the GA library [49, 50], manages distributed array data and provides

linear algebra operations, which the `TaoSolver` component (ANL) can use within advanced algorithms for unconstrained and bound constrained optimization problems; `TaoSolver` is built on top of the Toolkit for Advanced Optimization (TAO) [66, 67]. Under this project, additional linear algebra operations were added to GA as needed by TAO. As a benchmark and interoperability test, we ran a Lennard-Jones molecular dynamics optimization problem on the Dell Linux/IA32 and HP Linux/IA64 cluster at PNNL. The performance results indicated very good scaling (a simulation of 12,000 atoms yielded a speedup of 7.86 on 8 processors) and negligible overhead (less than one percent) when using the newly developed component interfaces that employ SIDL.

Furthermore, we are collaborating with chemists at PNNL and SNL to incorporate these components in the optimization of molecular geometries, which is a central function of quantum chemical calculations. To provide chemists with an unprecedented level of flexibility in determining molecular structures, we are using the CCA to link the NWChem [68] and MPQC [69, 70] quantum chemistry codes, which provide high-performance energy, gradient, and Hessian computations, with the `TaoSolver` optimization component as well as with the linear algebra capabilities within components built on the GA and PETSc libraries. Figure 2.1 depicts the components involved, where the components based on NWChem and MPQC are interchangeable, as are those based on GA and PETSc. Future work will refine the optimization and chemistry interfaces, examine performance issues to understand when the use of `TaoSolver` is preferred over existing functionality within NWChem and MPQC, and explore more complex optimization problems such as protein/ligand binding studies. See Section 2.4.2.b for further details.

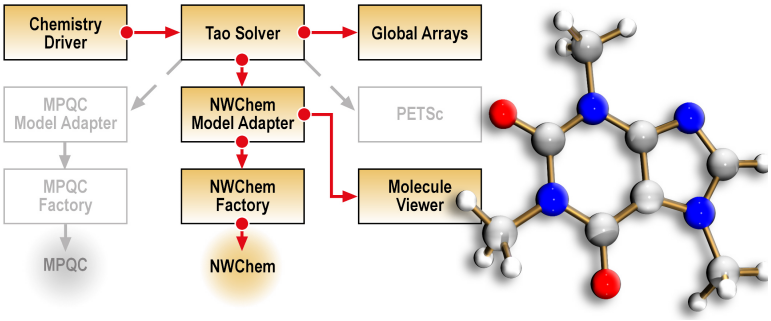

Figure 2.1: Component-based molecular geometry computations. Gray components can be swapped in to create new applications with different capabilities.

2.2.3.b Scientific Data Components in Combustion Applications

To demonstrate the feasibility and utility of using a component-based approach to scientific computing, we have developed several applications that solve PDE-based models and optimization problems. In the area of scientific data component development, our work with the Computational Facility for Reacting Flow Science (CFRFS) [3], a SciDAC application project, is of particular note. The overarching goal of this center is to develop a component-based toolkit for flame simulations, and the data component used in these simulations is based on the structured, adaptively refined meshes provided by the GrACE library [63]. As described in Appendix A and [71, 72], the component interfaces employ two primary abstractions: a GridHierarchy, which is a collection of hierarchical, nested Cartesian patches that provide increased mesh resolution in regions where simulation variables are changing rapidly, and DataObjects, which manipulate application field data.

The model of incorporating these data components with others for time integration as well chemical and physical models has worked extremely well for the combustion applications that are important to the CFRFS center. Already a number of component-based simulations of flame-like reaction-diffusion systems using numerical schemes with different accuracy and stability characteristics have been implemented. Runs on 32 processors are routine, and performance data shows that the component-based implementation has low overhead and is scalable [71, 72]. See Section 2.4.2.a for further details.

2.2.3.c Interactions with the TSTT, APDEC, and TOPS SciDAC Centers

In addition to working directly with applications groups, another primary mechanism for CCA impact on applications is through collaborations with numerical tool developers in the TSTT, APDEC, and TOPS ISICs to define common interfaces and component implementations for mesh management and discretization, as well as linear, nonlinear and optimization solvers. Our goal is that these new interfaces and components can eventually be employed by scientists who in turn directly collaborate with these mathematics centers.

For example, as discussed in Section 2.2.1.b, we have worked with domain experts in the TSTT and APDEC centers to define common interfaces for accessing mesh geometry and topology information. Prototype CCA-compliant implementations of the TSTT mesh interface have been used to solve simple PDE-based applications. These applications provide proof-of-principle demonstrations of the usefulness of components in scientific simulations and are used for teaching CCA concepts as part of our outreach tutorial efforts. In addition to developing simple applications that are compliant with common interfaces, we have created sophisticated hydrodynamics and combustion applications, as discussed in Section 2.2.3.b. This effort has enabled us to evaluate CCA technologies for performance and ease-of-use in complex PDE-based applications, both of which are critical for the acceptance of the CCA by DOE application scientists.

In the long term, the TSTT and APDEC common mesh interfaces and prototype implementations will be further developed to support sophisticated TSTT mesh management tools such as Overture (LLNL), NWGrid (PNL), and AOMD (RPI). When these tools comply with a common set of interfaces, they will be able to be used interchangeably in an application that is also interface compliant. This will enable application scientists to easily experiment with different mesh types to determine which is the best suited for the physics of a particular application. Related software such as discretization libraries, partitioning tools, and visualization tools will also become interface compliant, so that a suite of useful components will be available in a plug-and-play fashion.

We are also collaborating with colleagues in the TOPS ISIC to define interfaces for linear and nonlinear solvers, eigensolvers, and unconstrained and constrained optimizers. While this effort is more recent than the meshing and discretization projects discussed above, progress has already been made in exploring some early prototypes that use SIDL as the interface definition language. A goal of this work is to provide a simple approach for accessing the full range of TOPS tools that provide particular functionalities. For example, TOPS linear solver software includes the packages hypre (LLNL) [73], PETSc (ANL) [44], and SuperLU (LBNL) [74]. We plan to experiment with newly developed common interfaces to TOPS software in a variety of applications, including a SciDAC fusion simulation under development by the Center for Magnetic Reconnection Studies (CMRS, PI: A. Bhattacharjee) [75].

In addition, as an initial exercise in application code coupling, which is of strong importance to TOPS fusion collaborators in the CMRS and the SciDAC Center for Extended MHD Modeling (CEMM, PI: S. Jardin) [76], we plan to explore using the CCA to aid in the partitioning and coupled solution of a multi-component PDE model. We expect that this project will reveal issues that need to be addressed for multiple physics code coupling in more complex simulations.

2.2.4 Future Plans

Future research and development activities for scientific components are summarized in Table 3.2 (page 30). Through ongoing collaborations with other SciDAC mathematics and applications centers, we will continue to evaluate, extend, and refine the first-generation components developed thus far; see Appendix A for highlights of future plans for particular components. We also will develop new component functionality for multi-threading, load redistribution, computational steering, and fault tolerance. See the original CCTTSS proposal [15] for further details regarding this planned work.

Another new area of future work focuses on investigating quality of service (QoS) issues for numerical

components. Component technology facilitates the development of novel algorithms where choices among several functionally equivalent components can be made dynamically based on runtime conditions. For example, it might be possible to select at runtime among a variety of preconditioned Krylov methods based on problem size, memory availability, robustness requirements, etc. While component standards make it possible to, at least formally, re-use every possible implementation of a certain component type in a given program, mismatches could occur between the needs of a client component (for example, high accuracy) and the capabilities of a server component (for example, low accuracy). To support this dynamic behavior, an important research issue is the specification of the QoS characteristics required of and provided by components. We will develop mechanisms for specifying and predicting the QoS characteristics of parallel numerical components, where we define these as the accuracy, robustness, performance, and scalability of the component. See [77] for a discussion of preliminary QoS work. We will also collaborate with the PERC SciDAC center to investigate how QoS specification mechanisms can take advantage of PERC performance tools for runtime performance prediction and adaptation.

The CCTTSS plays an essential role in cross-cutting integration activities among other SciDAC groups by layering and adapting the work of other centers. In particular, we will continue to collaborate with other SciDAC teams to extend existing parallel software libraries to use component technology and to develop additional components. This component suite will provide basic functionality needed by a range of simulations, including combustion, fusion, chemistry, and climate, and will serve as a testbed for frameworks and related infrastructure. This component suite is in no way intended to provide the only implementation of any particular functionality. Rather, an important facet of our approach involves dialogue among the community to define domain-specific interfaces. Now is the time for such activities; multiple tools already exist, and we can exploit their differences and leverage their commonalties. We expect that this work will improve technology transfer among DOE laboratories and academia, since scientists will be able to incorporate the latest computational research methodologies if software provides standard hooks. In addition, academics should find it easier to incorporate DOE test problems to motivate their research and to evaluate the performance of newly developed algorithms.

2.3 Parallel Data Redistribution and Model Coupling

Coordinator: James A. Kohl (ORNL)
Investigators: David Bernholdt (ORNL), Randy Bramley (Indiana), Pat Fasel (LANL), Kate Keahey (ANL, formerly LANL), James Kohl (ORNL), Jay Larson (ANL), Sue Mniszewski (LANL), Steven Parker (Utah), Reid Rivenburgh (LANL).

A core problem faced by many high-performance scientific simulations is the coordination of distributed data among tasks executing in parallel. Data are often divided into subsets and scattered or copied across sets of parallel processes to improve communication patterns and locality of access by remote tasks. The details of this data distribution are often specially catered to each given algorithm. In a component-based simulation, the parallel data decomposition is further subdivided within the context of specific parallel components. Each parallel component functions as its own parallel program, and consists of a "cohort" of like component instances spread across the set of parallel SCMD (Single Component Multiple Data, the component analog of SPMD) processes. For parallel components to cooperate in a given simulation, the cohorts must be able to share and exchange parallel data. Such scenarios include coupled parallel models, data extraction for interactive visualization, and checkpointing for automated fault recovery.

We refer to this challenge as the "MxN" (pronounced "M by N") problem, indicating that a cohort running on "M" processors must share data with another cohort on "N" distinct processors, where M and N do not match in cardinality or topology. An example of this concept is illustrated in Figure 2.2. Each cohort's

data decomposition must be decoded to map data elements in one cohort with corresponding elements in the companion cohort. Because each parallel component can distribute data in a unique way such mappings can require complex redistribution operations and translations.

The CCA is developing generalized parallel data redistribution tools to alleviate the inherent complexity in composing parallel components. Generic instrumentation can describe each component's parallel data, and then be applied to transparently execute a variety of data exchange operations at run time. Beyond these fundamental parallel data redistribution capabilities, additional operations can be concatenated using additional component "filters" for spatial and temporal interpolation, unit conversions, etc.

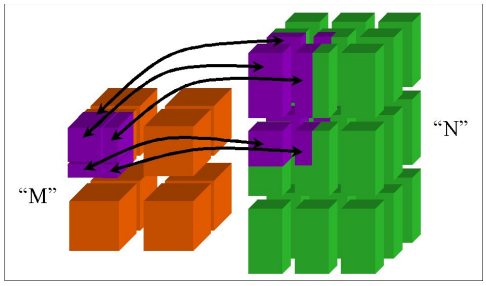

Figure 2.2: "MxN" Parallel Data Redistribution

Another related but distinct issue arises when sets of cooperating parallel components invoke methods on each other, referred to as Parallel Remote Method Invocation (PRMI). No well-defined semantics exist for the wide range of parallel invocation possibilities. Methods could be invoked with serial or parallel callers and callees, to perform coordinated parallel operations or else to independently update state in parallel. Such invocations could require data arguments or return results as either serial or parallel (decomposed) data arrangements.

Supporting PRMI is a problem unique to the CCA. Commercial component systems support only serial RMI, having no need for the added complications of massive parallelism and the SPMD model. The CCA programming model requires new semantics, policies, and conventions for invoking parallel methods and appropriately communicating function arguments and results. Synchronization is also a fundamental concern with PRMI, to ensure consistent invocation ordering and the coordination of parallel data arguments, as well as to avoid deadlocks and handle various failure modes.

2.3.1 MxN Parallel Data Exchange

Much progress has been made to date by the CCTTSS with respect to MxN technology, ranging from the development of generalized specifications to the implementation of multiple practical component-based MxN solutions. Two main existing software tools were applied to define and generalize the operations involved in performing parallel data redistribution – CUMULVS (ORNL) [59–62] and PAWS (LANL) [78]. PAWS is built on a "point-to-point" model of parallel data coupling, with matching "send" and "receive" methods on corresponding sides of a data connection. CUMULVS is designed for interactive visualization and computational steering, and so provides protocols for persistent parallel data channels with periodic transfers, using a variety of synchronization options. A generalized MxN specification has been developed that covers both of these connection models within a single unified interface.

The MxN interface includes several methods that define the key operations in performing parallel data exchange. Parallel components can *register* their parallel data by providing a handle to a "Distributed Array Descriptor" interface (see Section A.4). Registration provides the basic information about any distributed data decomposition and indicates the available access modes for MxN transfers. Parallel "communication schedules" can be defined that map elements from one data object to another. These schedules are then applied to define MxN "connections" using a variety of synchronization options. MxN connections can serve either "one-shot" transfers or persistent periodic transfers that proceed automatically every few iterations. Each independent portion of a transfer is initiated when an instance of the parallel cohort invokes the `dataReady()` method, indicating that local data is consistent and "ready" for the transfer.

MxN transfers can be initiated by either the source or destination components, or by a third party controller. Therefore, neither side of an MxN connection need be fully aware of the nature of any connections.

This situation greatly expedites the incorporation of existing parallel legacy codes into the component environment. Decisions about the connectivity of parallel data objects can be made dynamically at run-time, as no fundamental changes to the component code are necessary.

Based on this generalized specification, two distinct component implementations have been constructed, one based on each of the original tools (CUMULVS and PAWS). These components have been utilized in a variety of experimental component-based applications, with demonstrations at SC2001 and SC2002. Both the `CumulvsMxN` and `PawsMxN` components run single threaded. LANL's `PawsMxN` component, which exists as two pieces (sender and receiver), operates synchronously and provides parallel data transfer. A simple "ping-pong" demonstration of this component was made at SC2001. ORNL's `CumulvsMxN` component provides a discovery name service that allows components in the same or different frameworks to find each other and couple together using the MxN interface. `CumulvsMxN` has been primarily used to gather parallel data fields for visualization. Three different CCA component application demonstrations used `CumulvsMxN` at SC2001 [79], to display data based on rectangular meshes, unstructured rectangular meshes, and unstructured triangular meshes (using ORNL's `VizProxy` front-end viewer component, also based on CUMULVS). In addition, the University of Utah has developed experimental MxN component technology using SCIRun for visualization and computational steering.

2.3.1.a MxN and Combustion Simulations

In collaboration with the CFRFS [3] SciDAC project, work is currently in progress to utilize an extended `CumulvsMxN` component for coupling particle-based data fields to combustion simulations. This modified version of `CumulvsMxN` uses a `ParticleCollectionFactory` (see Section A.4) to describe a particle container and its encapsulated data fields. Note that fundamentally there is little in combustion simulations themselves that can benefit from MxN coupling, as the solution algorithms of the PDEs require that they work with local data with the barest minimum of ghost-cell updates. However, this view ignores adaptive chemistry and post-processing.

The bulk of the simulation time in combustion is spent on chemistry. A detailed chemical treatment is applied at every point, regardless of its necessity. One can adapt/simplify the chemistry depending upon the state at a point; further, once simplified, the chemistry does not change very quickly in time. The process of simplifying chemistry is complicated and time-consuming but only needs to be done infrequently. Further, post-processing of combustion data poses a few challenges. Most post-processing and computational geometry algorithms (for detecting structures in flame simulation solutions) are not very scalable; they can be made to work tolerably on a few CPUs, usually on SMPs with threading. Since the data sizes are large (and the algorithms expensive and not particularly scalable), it becomes imperative not to hold up the simulation's progress with post-processing and other computations that do not feed back into the simulation or else do so very infrequently.

Therefore, both adaptive chemistry and post-processing are good candidates for a quasi-offline treatment. The data from a simulation can be MxN-transferred to a smaller set of CPUs where the data are analyzed for reduction to simple chemical mechanisms or searched for structures. The bulk of the development of this computational singular perturbation (CSP, precursor to simplifying the chemistry) analysis work will be completed soon; the modified `CumulvsMxN` component will be applied to provide the input data. Work on structure-searching algorithms will take further exploration, but at this time the culminating particle-based CumulvsMxN component should satisfy the CSP demands.

2.3.1.b MxN and MCT/Climate

The climate community has vast experience in constructing parallel coupled models from individual parallel models. A prime example of this experience is the DOE-sponsored Model Coupling Toolkit (MCT) [80],

which has been supported by the SciDAC CCSM project [81]. MCT supports a variety of parallel execution modes for parallel coupled models, including sequential parallel coupling (event-loop) and coupling of asynchronous parallel models (either as a single or multiple executables). This work complements the generalized MxN technology by providing the specific details needed for coupling climate codes. We are currently working on re-packaging portions of MCT as CCA components, ultimately targeting a modified MxN interface specification. In the meantime, we will implement an MxN component on top of the existing MCT infrastructure.

MCT is similar to PAWS in that MxN transfers are implemented on a point-to-point model of parallel data coupling. MCT provides two types of parallel data transfer schemes. The first type is used for parallel data between concurrently running components (that is, each component resides on its own distinct pool of processors). Each component owns a communications scheduler called a Router, which contains all necessary information for the parallel data transfer. The information enables the source and destination components to appropriately pack, send, receive and unpack messages for the data transfers. The second MCT transfer scheme involves a parallel data redistribution operation for co-located SCMD components. In this scheme, the sending and receiving components reside on the same pool of processors, and the MxN transfer utilizes a communications scheduler called a Rearranger. The Rearranger comprises a pair of Routers (one for send operations, the other for receive operations), and sufficient information to avoid costly self-messaging.

We have also begun work on a strategy for packaging component models of a coupled climate model as CCA components. That is, we aim to package as CCA components individual atmosphere, ocean, sea-ice, land-surface, and river-runoff models as well as flux couplers. This effort faces a number of challenges: (1) creation of a workable interface between CCA and Fortran90, the language in which nearly all earth system models are written; (2) implementation of a component packaging scheme that is minimally intrusive; and (3) design of a scheme that is also compatible with other major componentization efforts such as the CCSM project's Model Coupling Toolkit and NASA's Earth System Modeling Framework [82].

Steps have been taken to address all three of the challenges stated above. The effort to create a Fortran-friendly interface has been described elsewhere (Section 2.4.2.c). A draft scheme for wrapping MCT components to re-cast them as CCA components also exists. Recent progress has been made in bridging the conceptual gap between MCT's programming model and ESMF's proposed "Superstructure" (also described in Section 2.4.2.c). Extensions to MCT have been implemented to follow the ESMF Superstructure classes of "Components," "Gridded Components," and "Couplers." In ESMF, a Component is an object that encapsulates all the data for a given model (e.g., an atmospheric general circulation model) as needed to couple to other models. A Gridded Component is a class encapsulating all the data used by a given model, including data needed for coupling. That is, a Gridded Component is a class built on top of the Component class. A Coupler encapsulates all the data needed for the transfer of data between two sequentially running components. New MCT objects that have been created to support this SCMD approach include the `mctComponent` and `mctCoupler` classes. An `mctComponent`, the MCT analogue for the ESMF Component, encapsulates all of a component's couplings to the outside world. An `mctCoupler` is the MCT analogue for the ESMF Coupler, and provides functionality above and beyond MxN data transfer by also supporting interpolation between physical meshes, as well as time averaging/accumulation of data. We have been able to re-implement the MCT SCMD unit tester using these new classes. A minor modification to remove the extra interpolation functionality from the `mctCoupler` will yield an MCT MxN component for the CCA that is suitable for SCMD data coupling.

2.3.1.c MxN Future Work — Implicit MxN Framework Solutions

While significant progress has been made in exploring and implementing initial prototypes for basic "MxN" parallel data redistribution, we have barely scratched the surface of the myriad of specific data organizations,

let alone the potential for more sophisticated application of this technology such as for model coupling tools. Work must be done to raise the level of abstraction from the low-level "assembly language" of parallel data exchange to provide more automated and intuitive interfaces and solutions for the typical scientific programmer.

Our preliminary investigations of high-level MxN data redistribution components reside at the application level, above the base framework. This approach has the most flexibility and precise semantics for exerting explicit control over various redistribution functions. Moreover, this approach is extensible because additional components can be independently developed and instantiated as needed without modification of the framework or the base interface specification. The underlying framework implementation is also simpler and more general, and does not include any hard-wired solutions for parallel data exchange or method invocation. However, the user must take responsibility for understanding and invoking the necessary data redistribution methods. This favors more sophisticated users over non-experts.

A related approach involves the design of low-level framework services that support *implicit* parallel data redistribution. This approach hides many of the details from the user with more "automatic" handling, but increases the cost in terms of framework complexity and loss of generality. Various choices for redistribution functions are registered with the framework at configuration time, or built into individual component implementations. These functions are automatically invoked when parallel data movement is indicated, whether triggered by parallel component connections or when parallel data objects are passed as arguments to method invocations. However, more burden is placed on component and framework developers to supply the necessary redistribution hooks and implementations.

We will explore the development of a multiplexer (MUX) service for maintaining the redistribution interfaces available for each component. The MUX will generate routing tables and schedules for data redistribution, and will handle any remote parallel data holder creation and destruction. The MUX will intercede and reconcile parallel data arguments against the user's data layout at method invocation time. The MUX will also convert any results of the invocation back into the original data layout.

Ultimately, these two MxN approaches (component-based and framework MUX service) can be *combined* given some enhancements to the basic CCA framework services model. The implicit framework service could use the explicit component-based implementations by applying a new pluggable framework service interface. This interface, now under development, allows a component to register its methods within a framework, so that components can use them without an explicit port connection.

2.3.2 Parallel Remote Method Invocation

An interface for Parallel Remote Method Invocation (PRMI) is essential for any framework that supports interactions among parallel components. Yet a policy is required to define the expected invocation behavior, especially for partial execution or failures. Methods must accept both scalar and parallel data as input arguments and return values. Appropriate semantics are needed to interpret such uses, e.g., whether scalar method arguments are copied to every cooperating thread or sent only to some designated one. Coordinating the synchronization of invocations is also important to avoid potential pitfalls such as deadlock.

In 2001, in collaboration with Kate Keahey (formerly at LANL, now ANL), PAWS technology was used to build a prototype system for parallel remote method invocation (PRMI). Policy issues for PRMI were identified, such as invocation scheduling on remote processes. PRMI was demonstrated as a composition of MxN transfers for marshaling parallel data arguments, allowing invocation similar to local method calls (e.g., `component.foo(A,B)`). Additionally, the use of multi-function MxN components was shown to be a desirable optimization, based on a "flip and MxN" example (flipping data on one dimension in conjunction with a transfer).

Next, in conjunction with NSF-funded research, another prototype PRMI system was developed using parallel data redistribution. This prototype provides semantics similar to a method call and semi-

transparently redistributes data between the calling set of processors and the callees. Due to the variety and complexity of the potential PRMI semantics and scenarios, this prototype provides just two basic types of parallel RMI semantics: *collective* and *independent*. It is our belief that these two options, combined with an intra-component programming model such as PVM [26] or MPI [25], will cover most of the reasonable scenarios.

A PRMI call with *collective* semantics requires that all of the caller processors participate in the call and ensures that all "callees" get called exactly once. This requirement is met via an underlying PRMI framework service that manages execution of the method invocation, gathering or scattering the control flow as needed. This scheme works whether there are more callers than callees, or vice-versa. The real value of these collective calls is seen when coupled with a redistribution mechanism for moving parallel data back and forth in an organized manner. *Independent* calls require less synchronization and match a single call from one of the caller processes to one of the callee processes. As such, the underlying framework service need only choose, in some arbitrary and efficient manner, which callee task will process any given invocation request. This scenario is amenable to certain parallel client-server applications, wherein multiple callee "server" processes can handle a series of independent requests from a client collection.

We have also developed an experimental extension to SIDL that facilitates parallel array redistribution by defining the nature of the data expected by method arguments. In addition to these SIDL modifications, new methods were created to exchange data distribution information from each process at run time. Combined with the PRMI functionality above, this provides a transparent and flexible mechanism for exchanging large scientific datasets in parallel method invocations.

2.3.2.a PRMI Future Work — Transport Mechanisms

In addition to the ongoing fundamental research into PRMI interfaces and system design, we will also explore some underlying transport mechanisms for generalizing the actual exchange of data arguments. High-performance encodings will be applied to enable interoperability with industry systems. For example, SOAP is an emerging industry standard for interoperable RMI encoding. SOAP encodes remote method invocations using XML, and transports them via generic protocols such as HTTP or SMTP. Unfortunately, SOAP does not efficiently encode large blocks of raw data [83]. We will investigate extensions to SOAP that add optimizations for efficient terascale data transport.

2.3.3 Distributed MxN

While a significant proportion of scientific simulations would utilize the parallel SCMD paradigm, a large body of distributed computing applications can also benefit from MxN and PRMI technology. For example, large distributed sensor networks can produce an immense amount of information that must be collected for central processing.

Part of Indiana University's contribution has been the exploration of a mechanism for distributed MxN technology. An initial prototype uses communication based on the stream paradigm, where data is logically transfered between the participating components as if through a stream or a file (although the data is transfered in parallel and is never explicitly serialized).

The main design goal for this system has been to ease the migration of existing applications (with file-based I/O interfaces) into CCA components, forming larger multiphysics simulations. Using the standard MPI I/O system for exchanging data in parallel codes is the analog to file-based exchanges in serial codes. With minimum changes, applications can use this variation of the MxN system to communicate in real time with other applications. This approach also allows unit testing to be performed by allowing the application to switch seamlessly between live MxN I/O and file I/O for testing. Tests have shown that this approach is

significantly faster than either of the usual options for quickly connecting two application codes (writing to files that are then shared, or connecting the rank 0 processes for each code) [84].

2.3.3.a Distributed MxN Future Work — Full Prototypes

A current test prototype of this technology is based on the ROMIO MPI-I/O implementation [85], which underlies MPICH and LAM MPI. Integration with MPI allows the system to understand any MPI derived data type, a significant advantage of this approach. We will be conducting scalability tests of this "MxN device" on recently acquired large clusters (two at 200 CPUs each). The current interface to this system is through the regular MPI-I/O API, and we are considering encapsulating this functionality in a new distributed MxN component that mediates the data transfers between two MPI programs. We are considering integrating this technology into the Proteus [86] protocol as well.

2.3.4 Future Research — Model Coupling Technology

Exchanging elements from parallel or distributed data structures is merely the beginning of true technology for parallel model coupling and data sharing. Depending on the nature of the actual data structures involved, significant data translations could be needed beyond the simple MxN mapping of data elements. If the source and destination data use different meshes or spatial coordinate representations, or are computed in different units or at different time frames, then several additional data translation and conversion components will be required to fully transform and share semantically comparable parallel data.

A wealth of interpolation and sampling schemes are available for translating data among desired spatial or temporal formats. Historically, such schemes carry with them an almost religious stigma, and there is much debate among scientists on the merits of one scheme over another. We will extend our collection of interface specifications to include hooks for supporting generic data transformations and conversions. Given sufficient flexibility in the arguments for these interfaces, a wide range of implementations can be built to cover any relevant interpolation or conversion algorithm.

To utilize the resulting sequence of data transformations and data redistributions, a "pipeline" of components must be assembled. An important pragmatic issue that arises with such pipelining is how efficiently redistribution functions "compose" with each other. Techniques will be explored to operate on data "in place" and avoid unnecessary data copies. "Super-component" solutions will also be explored for some common cases by combining several successive redistribution and translation components into a single optimized component.

2.4 User Outreach and Applications Integration

Coordinator: David E. Bernholdt (ORNL)

Like the other SciDAC ISICs, the CCTTSS combines traditional research with an explicit effort to provide production quality tools and technologies for use by SciDAC applications groups and by the broader scientific computing community. The User Outreach and Application Integration thrust of the CCTTSS R&D program is intended to facilitate adoption of the Common Component Architecture, to help insure a feedback loop between CCA adopters and CCA developers, and to help us develop "best practices" for the use of components in large-scale high-performance scientific simulation software.

As described in more detail below, our outreach work includes both general educational activities and interactions with specific groups around scientific applications in which they want to use the CCA. Our educational efforts include papers, conference presentations, tutorials, and related activities designed to inform the computational science and computer science communities about the component-based software

engineering and the CCA. Our work with applications groups ranges from tight collaborations (often where at least one researcher has strong ties to both the CCTTSS and to the application group) to looser consultative relationships.

2.4.1 Education and General Outreach

2.4.1.a Tutorials

CCA Forum Tutorial Working Group: Rob Armstrong (SNL), David Bernholdt (ORNL, chair), Lori Freitag Diachin (SNL, previously ANL), Wael Elwasif (ORNL), Dan Katz (JPL), Jim Kohl (ORNL), Gary Kumfert (LLNL), Lois Curfman McInnes (ANL), Boyana Norris (ANL), Craig Rasmussen (LANL), Jaideep Ray (SNL), Torsten Wilde (ORNL)

Our most significant effort in the Education category has been the development and presentation of a tutorial on the CCA. After presenting a number of more or less off-the-cuff tutorials during the autumn of 2001, we recognized the need for an organized effort to develop a common base of material that we could share and present in a variety of contexts.

Building from the first multi-presenter CCA tutorial, in conjunction with the CCA Forum Winter 2002 meeting, the CCA Forum established a Tutorial Working Group. Through the extensive efforts of Working Group members, the tutorial has evolved into a set of eight modules covering the philosophy and background behind the component concept and the CCA, how to create and use components (including live examples), and a look at how CCA users are building scientific applications. The tutorial is presented in either a six hour full-day format, or a condensed four hour half-day format, depending on the situation. Both the material and presentation style of the tutorial have evolved continuously since its inception, reflecting the evolution of the CCA tools and our thinking, and in response to feedback from tutorial attendees, many of whom have attended several times. Following a suggestion by Kate Keahey (ANL), an early participant in the CCA Forum, we typically present tutorials in conjunction with CCA Forum quarterly meetings. This approach gives newcomers and (prospective) users a chance to learn more about the CCA.

In addition to the five tutorials at CCA Forum meetings in 2002-3 to date, we have had the opportunity to offer three additional tutorials:

- ACTS Collection Workshop at LBNL, September 2002,
- Los Alamos Computer Science Institute (LACSI) Symposium in Santa Fe, NM, October 2002, and
- SC2002 in Baltimore, MD, November 2002.

The lengthy selection process and visibility of the SC2002 tutorial made this event the culmination of our tutorial development efforts over the preceding year, and we were very pleased with the results.

We believe that the tutorials are a very effective way to reach beginning users, and we also make the presentation materials and software examples from the tutorial available on the `http://www.cca-forum.org` web site for self-study. We also have a standing offer to give "customized" tutorials for applications groups, (domain-oriented) conferences, and other venues, but other than the ACTS Collection Workshop and LACSI, we have received no requests to date (the SC2002 tutorial was undertaken on our own initiative). This reflects probably our only significant disappointment regarding our tutorial efforts to date: that a relatively small portion of the potential user community has taken advantage of them. This may be a simple matter of communications. While we encourage CCTTSS researchers in the areas where tutorials are taking place to "beat the bushes" among their SciDAC (and other) application contacts, we do not have a simple, general way to reach all other SciDAC PIs of such events.

2.4.1.b Publications, Presentations, and Other Activities

Since its inception, Center members have published more than 80 presentations and papers related to the project. These are cataloged in Appendix B. Appendix C catalogs additional outreach activities, including presentations and demonstrations in less formal venues, such as the SC conference exhibition hall, and informal interactions with (prospective) users. This list is certainly incomplete, due to the informal nature of these events, and the fact that we do not track them so rigorously, but it is a representative subset of our activities.

2.4.2 Applications

Although, as part of the SciDAC program, the CCTTSS is a relatively new effort, we are fortunate to have been able to build on the earlier efforts of the CCA Forum, which included prototypes of many of the tools we're developing, and relatively mature specifications for the component architecture itself. Consequently, though the tools and even the specification are still evolving, they are sufficiently mature and stable to allow us at this early stage of the project not only to talk to applications groups about how to plan for and migrate to the CCA, but also to work with a number of early adopters to produce actual CCA-based applications. Here, we highlight a few efforts in which CCA integration is the most advanced.

The investigators are mentioned in conjunction with each effort are those most directly involved in the specific CCA integration effort. Clearly all of these efforts build on extensive contributions of others, too numerous to mention, without whom these CCA integrations efforts would have been impossible.

2.4.2.a Computational Facility for Reacting Flow Science

Investigators: Sophia Lefantzi (SNL), Jaideep Ray (SNL), Sameer Shende (Oregon)

The Computational Facility for Reacting Flow Science (CFRFS, PI: H. Najm, URL: `http://cfrfs.ca.sandia.gov`) is supported by the SciDAC Basic Energy Sciences program. The project envisions a computational facility for flame simulation where combustion and computational researchers can implement physical and chemical models as well as numerical algorithms with a minimal knowledge of the supporting infrastructure. A key figure in the adoption of CCA in this project has been Jaideep Ray (SNL), who receives support from both CFRFS and CCTTSS, and has been involved with the CCA effort for several years.

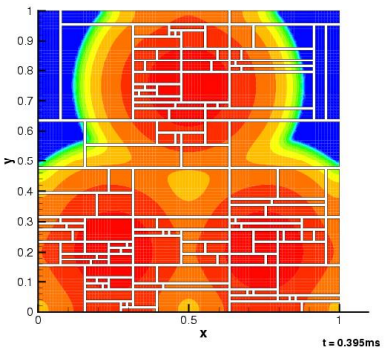

Figure 2.3: Temperature distribution 0.395 ms after inception of heating. The white lines denote domain decomposition across 28 CPUs.

Simulations of flame-like reaction-diffusion systems with replaceable models have been made possible through the use of CCA components for time integration, structured adaptively refined meshes (SAMR), and physical models. Figure 1.2 (page 3) show the "wiring diagram" for the CCA-based application and Figure 2.3 shows the evolution of ignition fronts in an igniting H_2-Air mixture.

Within four months, CFRFS researchers incorporated a second generation of components that embody higher accuracy and stabilized numerical techniques by replacing just a few components in the application. The CCA has also been instrumental in facilitating the use of external software, including components from the SciDAC PERC and TOPS Centers (based on the TAU performance tool and the CVODES integrator, respectively).

Forthcoming work includes developing new convective physics capabilities as well as new SAMR numerical schemes to achieve complete flame simulations.

2.4.2.b Computational Chemistry

Investigators: Steve Benson (ANL), Ronald Duchovic (Indiana-Purdue Fort Wayne), Wael Elwasif (ORNL), Curtis Janssen (SNL), Liz Jurris (PNNL), Manoj Krishnan (PNNL), Lois Curfman McInnes (ANL), Jarek Nieplocha (PNNL), Boyana Norris (ANL), Craig Rasmussem (LANL), Jason Sarich (ANL), Theresa Windus (PNNL)

As noted in our proposal, computational chemistry is fundamental to DOE's science mission and the size and complexity of software and problems in this domain offer a particular opportunity. The software in this domain is dominated by a relatively small number of well-established, large, complex packages, which are not designed for easy interoperability. Two efforts are currently underway in the computational chemistry area aimed at demonstrating increased interoperability through the CCA.

The major effort is the "Application Focus" in computational chemistry described in our proposal. The two application focus projects (in computational chemistry and climate modeling) were designed, in part, to help insure a tight feedback loop between application developers and CCA developers and also to examine issues around the use of existing large-scale code bases in a CCA environment, including the development and implementation of common interfaces.

Historically, chemists have tended to develop fairly monolithic code, with limited use of external libraries. Consequently, it can be difficult for computational chemistry packages to take advantage of the state of the art in, for example, linear algebra algorithms, including the optimization routines used to determine molecular structure. At the same time, the problems being treated are increasingly challenging to traditional optimization algorithms.

Our work to date focuses on the molecular structure optimization capabilities, which are central to much of the science done with these codes. To increase interoperability and generality, we are replacing the internal optimization capabilities of two quantum chemistry codes, NWChem (PNNL) [68] and MPQC (SNL) [69, 70], with the TAO optimization package (ANL) [66, 67], which is affiliated with the SciDAC TOPS center. We have designed a component interface for the evaluation of the molecular energy and gradient, which have been implemented in NWChem and MPQC. Through a simple adapter, this interface is compatible with the component interface designed for TAO. This makes it possible to switch the quantum chemistry package (i.e., between MPQC and NWChem) in the middle of an optimization – a practical impossibility in implementations built using traditional approaches. TAO's linear algebra needs are also expressed through a component interface, for which implementations based on both PETSc (ANL) [44, 45] and Global Arrays (PNNL) [49–51] exist. The components involved are shown in Figure 2.1 (page 14).

A further benefit of component-based software engineering is also clear in this work. The teams at ANL, PNNL, and SNL were able to collaborate in the development of a component-based application by merely focusing on the interfaces and their own components, without the need to work on unfamiliar code developed by the other groups.

Future plans include extending the optimization application to more complex problems such as protein/ligand binding studies, and looking a deeper levels of interoperability between computational chemistry packages.

We are also collaborating with the Advanced Software for the Calculation of Thermochemistry, Kinetics, and Dynamics project (ASCTKD, PI: Al Wagner), part of the SciDAC Basic Energy Sciences program, to develop component-based software to study reaction dynamics. The first step in this process involves work on POTLIB [87, 88], which provides a library of analytically modeled potential energy surfaces for numerous reactions. Our work to date has focused on developing a component interface to POTLIB. The effort required here has been more significant than for most of the other interfaces we have helped to design and implement because the original POTLIB interface was implemented in Fortran and makes extensive use of globally-visible common blocks, a technique that goes against the basic concepts of component-based

software engineering. Nevertheless, work is proceeding.

Component-related work in the ASCTKD effort is being carried out primarily within the project itself, with a number of CCTTSS researchers acting in primarily consultative and educational roles. In keeping with its being a CCTTSS Application Focus, the optimization effort is being carried predominantly by CCTTSS-funded researchers, with additional contributions to the TAO side of the effort from the SciDAC TOPS center.

2.4.2.c Climate Modeling

Investigators: John Drake (ORNL), Wael Elwasif (ORNL), Michael Ham (ORNL), Jay Larson (ANL), Everest Ong (ANL)

Computational climate modeling is critically important for our understanding of global processes and the potential for human impact. CCTTSS is collaborating with two major climate modeling efforts in a two-way exchange. The Community Climate System Model (CCSM) [81] is a major national effort in high-end parallel climate modeling, supported by the SciDAC Biological and Environmental Research program and other sponsors. The Earth System Modeling Framework (ESMF) [82] is a NASA-sponsored effort to develop a software framework to support and facilitate the future development of climate modeling software, promoting reuse an interoperability within the community. Like the computational chemistry effort, climate modeling was identified in our proposal as an Application Focus within CCTTSS, meaning that the Center uses some of its funding to advance the use of components in this area. Our emphasis on these two projects is due to special opportunities that they present: a two-way exchange of information and software for model coupling with the CCSM, and the timeliness of NASA's commissioning of a domain-specific framework, which is clearly a candidate for use of component-based software engineering.

As with the CFRFS effort, our work in the climate community is anchored by researchers with ties to two or all three of the groups involved: Jay Larson (ANL) is associated with CCSM, CCTTSS, and ESMF; Michael Ham (ORNL) is associated with both CCSM and CCTTSS.

The CCSM offers several different levels of granularity for the introduction of components. They can be used to link together models (i.e., atmosphere, ocean, etc.), at the parameterization level within models, and at the algorithmic level. The order in which these are listed reflects the order in which we anticipate the traditionally conservative climate community will accept introduction of this new software paradigm. Our collaboration with the CCSM involves interactions at all three levels.

At both the coarsest and finest levels, the work focuses on the DOE-sponsored Model Coupling Toolkit (MCT, ANL, Figure 2.4) [80]. This toolkit embodies the climate community's experience with constructing parallel coupled climate models from individual parallel models, which is valuable to the CCTTSS effort in MxN Parallel Data Redistribution to support generalized model coupling. Jay Larson and others associated with the MCT have already been active in the initial CCTTSS MxN development efforts, and we plan a deeper exchange of experience and tools as MCT-based components are developed. This componentization work is already underway, with an initial focus on the basic data types and computational

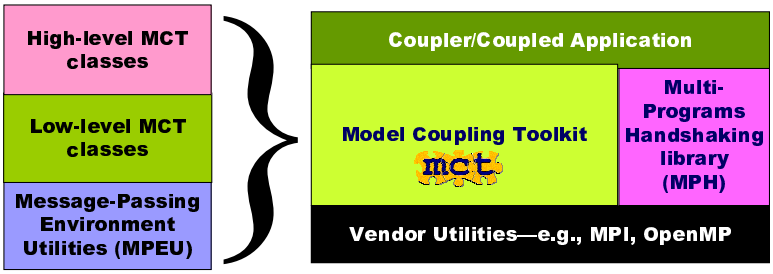

Figure 2.4: The general architecture of the Model Coupling Toolkit and its relationship to the computational environment (supporting libraries and the application using MCT).

tools required to create simple couplers.

At the algorithmic level, the CCA has the potential of accelerating the development process of individual climate physics parameterization components and dynamical core components. As a first demonstration of this potential, we are developing a new river runoff model built primarily from CCA components for matrix-vector multiplication derived from the MCT.

At the physics parameterization level, we envision two major applications: (1) componentization of major software entities in models, and (2) componentization of individual subgridscale physics parameterization packages. In conjunction with a CCSM internal effort to refactor the Community Atmosphere Model (CAM) to better separate the "physics" and "dynamics" aspects of the model, we are investigating casting them as CCA components. Casting these major portions of CAM as components would allow one to replace the dynamical core of the CAM with a new one with relative ease. At the individual parameterization package scale, we envision componentizing the individual schemes (e.g., cumulus parameterization) that collectively comprise the model's physics. Success in these efforts could have a profound impact on how model development is done by speeding up the validation process of dynamical cores and physics parameterizations. It could also change dramatically the reporting of progress in these areas if scientists not only publish their results, but publish their software as CCA components.

The ESMF is primarily a project to develop a standardized computational infrastructure upon which a wide range of climate models and related tools might be implemented (see Figure 2.5). The CCSM is one of the many climate efforts expected to adopt the ESMF's tools. While the ESMF's own terminology treats the climate models as "components" which plug into a relatively heavy-weight "framework" or "infrastructure," it is quite logical to envision from a CCA viewpoint that the framework itself, and the controlling "superstructure," in addition to the climate models themselves, could be implemented as (collections of) CCA components. Our work with the ESMF is focused first on insuring compatibility with CCA concepts and tools as the ESMF strives to meet very specific software milestones on a tight three-year timeline. We are also collaborating to produce practical demonstrations of CCA-based versions of elements of the ESMF infrastructure. These include the basic "field" data type, which will be used in the CCSM CAM componentization effort, coupling-related capabilities (the ESMF coupling facility will be derived from MCT), and the "superstructure" which orchestrates coupled computations.

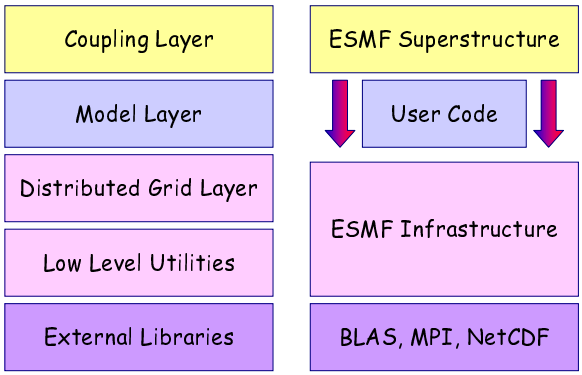

Figure 2.5: The general architecture of the Earth System Modeling Framework. A functional breakdown is shown on the left, while the ESMF's terminology is shown on the right. The ESMF thinks of the "User Code" as pluggable components, while they provide the superstructure and infrastructure.

It is important to acknowledge that our progress with the climate community has been slowed by the lack of support within CCA for Fortran90 (and successor standards). This was the first community we encountered with both a serious interest in the CCA and "serious" use of F90 features. As such, our interactions with them have been instrumental in reshaping the Center's approach to F90 support (see Section 2.1.2). Progress with both Babel and Chasm has recently made it feasible to begin serious work on the efforts described above. Chasm has also been instrumental in helping to develop a strategy for componentization that is minimally intrusive on existing Fortran code – another important consideration for acceptance of component-based software engineering in this field.

2.4.3 Future Plans

2.4.3.a Education and General Outreach

We believe that the development of the tutorial was a critical and very successful element for our early outreach efforts. We do plan to continue to evolve the tutorial, and to present it wherever opportunities arise. In order to try to reach a broader audience, we plan to target more conferences, and we will also investigate making tutorial presentations via the Access Grid. However, overall, the tutorial will be reduced in importance as we shift our attention to other important outreach areas.

Our next "Education" focus area will be on written documents, both documentation relating to CCA tools and their use (including documenting "best practices") and technical papers describing CCA research. One of our first priorities will be a general CCA overview paper to update the 1999 HPDC paper [1], which is now rather dated. With respect to documentation of CCA tools and techniques, our first priority will be to develop a written complement to the tutorial, a kind of "beginner's guide" to writing and using CCA components.

2.4.3.b Applications

A significant number of application groups have expressed their intent to adopt the CCA but have not yet progressed much beyond initial discussions and planning. Considering that the CCA effort is still in a relatively early stage, and that many prospective users have had to make a significant initial investment in organizing their efforts internally and were not ready to take on significant new software technology, we find the current situation neither surprising nor troubling.

At the same time, we have anecdotal evidence that the Common Component Architecture is gaining recognition and use, and among our outreach activities are presentations to or meetings with quite a number of prospective CCA users. In addition, we have always expected that the primary driver for CCA adoption for many groups will be the plans of the SciDAC Applied Math ISICs, with whom we collaborate closely (see Section 2.2.3.c), to deploy their tools in CCA form.

We believe that the initial success with applications in combustion and chemistry, as well as the developing climate effort, will be important factors in helping to convince other application groups that the CCA is ready for "serious" use. Consequently, in the coming year, we expect to see a change in the number of applications actually creating CCA components, as well as a change in the dominant mode of interaction with these groups, leaning toward more of a consultative approach.

3. Deliverables

The following four tables are revised deliverable schedules that directly correspond to four tables in our original proposal [15, pp. 23–25]. The column headings were changed to avoid confusion. The original proposal had column headings of '01–'05, but SciDAC funding and effort did not begin until July, 2001.

Table 3.1: **Frameworks Deliverables.**

July 2002	July 2003	July 2004	July 2005	July 2006
Complete candidate SCMD implementation (SNL)	Adapt strict SCMD concepts to more general scheme for HPC (SNL)	Improve dynamic user interaction of SCMD framework (SNL)	Iterate on design & improve per user requirements (SNL)	Iterate on design & improve per user requirements (SNL)
Complete candidate distributed computing CCA runtime (IU)	Integrate Microsoft .NET components; release initial library of Grid Service components (IU)	Release beta version of integrated framework to application teams (IU, SNL)		Release final public version of integrated framework (IU,SNL)
		Develop XML schema for access to Alexandria repository (LLNL, SNL, IU)	Add support for remote access by component framework tools to Alexandria repository	Deploy Alexandria component repository & assist collaborators with integrating components
Add Python & client-side Java to Babel(LLNL)	Open Babel & Add Fortran90 (Phase I) to Babel (LLNL)	Add server-side Java to Babel, continue Fortran90 improvements (LLNL)	Add MATLAB to Babel, continue Fortran90 improvements (LLNL)	Add Babel support for COM & .NET for Windows (LLNL)
Complete basic concurrency standard & implementation (all)	Evolve concurrency design per user requests	Evolve concurrency design per user requests	Evolve concurrency design per user requests	
		Develop a distributed multiplexer & XML protocol prototype (IU, LLNL, SNL)	Iterate on SCMD/Grid multiplexer per user requirements (IU, SNL)	

Notes: LLNL team deferred additional server-side Java and Alexandria effort to concentrate on Fortran90. Fortran support in general has become the top priority for Babel.

Table 3.2: **Scientific Components Deliverables.**

July 2002	July 2003	July 2004	July 2005	July 2006
Evaluate application requirements (all)	Deploy prototypes, evaluate & refine with apps feedback (all)	Evaluate, refine, & extend components; offer tutorials (all)	Evaluate, refine, & extend components; offer tutorials (all)	Evaluate, refine, & extend components; offer tutorials (all)
Develop prototype components for 2D/3D visualization (ORNL)	Deploy prototype 2D/3D visualization components as Linux RPMs (ORNL)	Construct and deploy SIDL-based 2D/3D visualization components (ORNL)	Create extended viz components for simple parallel rendering (ORNL)	Extend viz support to desktop (ORNL, UU)
	Develop prototype computational steering components (ORNL)	Deploy prototype computational steering components as Linux RPMs	Add port fault notification & hooks into SCMD event service (ORNL, SNL)	Develop prelim SCMD fault monitoring & recovery component (ORNL, SNL)
Develop prototypes for GUI components (ANL, UU)	Develop prototypes for load redistribution and multi-threading components (SNL)	Use GUI components for viz & numerical computations (ANL, ORNL, UU)	Use load redistribution and multi-threading components in chem apps (SNL, PNNL)	
Develop prototype components for linear & nonlinear solvers, optimization, & low-level services (ANL, IU, with TOPS ISIC)	Define interfaces between MPQC, NWChem, & TAO (ANL, PNNL, SNL); develop prototype component factory for Jacobians/Hessians (ANL)	Investigate app-specific components in optimization (ANL, PNNL, SNL); develop models for composing QoS characteristics (ANL)	Develop interfaces for multilevel nonlinear solvers (ANL, with TOPS center); integrate QoS system into repository (ANL, LLNL)	Evaluate components using QoS mechanisms in various apps (ANL, PNNL, SNL)
Define interfaces for local raw data, distrib. & dense arrays, meshes & fields (ANL, ORNL, PNNL, SNL)	Define interfaces for sparse arrays & global ops; release preliminary spec for scientific data component (ANL, ORNL, PNNL, SNL)	Define interfaces for app control of mesh modifications (ANL, ORNL, PNNL, SNL, with TSTT ISIC)	Circulate new data specs; deploy components supporting adaptivity (ANL, ORNL, PNNL, SNL, with TSTT ISIC)	Add support for hybrid schemes (ANL, ORNL, PNNL, SNL, with TSTT ISIC)
Develop prototype SAMR component (SN)	Deploy SAMR component in combustion app (SNL)	Perform initial high-fidelity 3D combustion runs using SAMR component (SNL)	Test SAMR component wth complex flame simulation (SNL)	

Notes: Because of emphasis within the CCTTSS on promoting common interfaces for numerical and application domains, the scientific components effort will concentrate on this work, thereby putting off development of multithreaded components to year 2, and delaying the data interface broker work indefinitely. A CCA-compliant prototype GUI component has also been developed at ANL that will function in all CCA frameworks.

Table 3.3: **Parallel Data Redistribution Deliverables.**

July 2002	July 2003	July 2004	July 2005	July 2006
Develop prototype MxN component for structured meshes based on CUMULVS (ORNL, SNL)	Deploy prototype MxN components as Linux RPMs (ORNL)	Construct and deploy SIDL-based MxN components (ORNL)	Develop generalized MxN parallel data exchange component (ORNL)	Develop preliminary data translation and interpolation components for model coupling (ORNL, SNL)
Develop prototype MxN component for structured meshes based on PAWS (LANL)		Develop and deploy particle-based MxN components (ORNL)	Integrate ARMCI into MxN parallel data exchange component (ORNL, PNNL)	Experiment with optimized composite data redistribution components (ORNL)
Investigate preliminary programming models for PRMI (LANL, LLNL, UU)	Investigate SOAP performance issues & improve performance for data redistribution (IU)	Add RMI to Babel (LLNL, IU)	Add PRMI to Babel language interoperability tool (LLNL, IU, UU)	Extend PRMI system to support parallel data arguments (LLNL, IU, UU)

Notes: Because MxN development has centered on incorporating Babel into the existing components, we have deferred work on better responding to user requests, fault tolerance, ARMCI, and visualization extensions for MxN. The MCT will be contributing MxN functionality to the CCSM because it also forms a major component of the ESMF, an important initiative in the climate community.

Table 3.4: **Application Integration Deliverables.**

July 2002	July 2003	July 2004	July 2005	July 2006
Liaison (especially education and needs assessment) with outside projects (ORNL lead)	On-going liaison work with outside projects (ORNL lead, all participate)	On-going liaison work with outside projects (ORNL lead, all participate)	On-going liaison work with outside projects (ORNL lead, all participate)	On-going liaison work with outside projects (ORNL lead, all participate)
Define prototype molecular energy interface (ORNL, PNNL, SNL); develop molecular energy component adapters for MPQC (SNL) and NWChem (PNNL)	Extend molecular energy interface & update component adaptors (ORNL, PNNL, SNL); demonstrate integrated TAO/electronic structure app (ANL, PNNL, SNL)	Define protein/ligand binding interface (SNL)	Develop and demonstrate one-electron property interface (PNNL, SNL)	Develop and demonstrate solvation interface (Ames, PNNL, SNL)
	Develop prototype DIRDY interface (PNNL)	Demonstrate integrated DIRDY / electronic structure app (PNNL)	Demonstrate advanced kinetics application (ANL, PNNL, SNL)	
Work with data components and MxN redistribution groups to produce standard interfaces (ANL, ORNL)	Package MCT MxN as a CCA component; examine interoperability between MCT and other MxN implementations (ANL, ORNL)	Demonstrate component-based and atmosphere models (ANL, ORNL)	Evaluate and tune CCA-MCT and MxN components; expand CCA beyond CAM, MCT (e.g., model diagnostics, data assimilation) (ANL, ORNL)	Demonstration of CCA-based coupled climate model (ANL, ORNL)

Notes: The timeline and specifics of the climate activity have been adjusted to reflect a later start due to F90 issues and changes within the CCSM and ESMF projects with whom we are collaborating.

4. Closing Perspective

The CCTTSS is based intentionally on the work begun and ongoing in the CCA Forum and is devoted to making high-performance components a reality. Within the center, investigator interactions are centered around the open source software development model that has been so successful in the Linux community. This approach is a constant that will not change. Almost everything else is flexible, and as investigators run into blocks or have bursts of insight, course corrections will be made. From an overall perspective, the largest change since instituting the CCTTSS as a SciDAC center has been a movement from components as theory, to plug-and-play software that application builders use. Every course correction has been geared to delivering tools, education, and directly helping SciDAC investigators outside the CCTTSS transition into the high-performance component world. Even the research work has become centered on making components more painless, automatic, and more interoperable.

A few examples are in order. Babel began as a LLNL-only team with an implementation and some in-house users. As Babel gained acceptance, it also became part of many collaborators' critical paths. As more people wanted to contribute to these ideas at many levels, we sought ways for Babel development to become more participative. This forced some hard thinking about what is essential for interoperability, and what is a matter of taste. As it developed, the triumvirate of the SIDL grammar, its XML equivalent, and the C-based Intermediate Object Representation (IOR) was determined to be the irreducible core of Babel; a modification in one commonly cascades changes to all. From that Core came the notions of Babel Extensions and Babel Dialects. Extensions are "plug ins" that augment Babel without violating its interoperability guarantee. All of Babel's existing language bindings are, from this perspective, extensions. Babel Dialects, while sometimes necessary for exploratory research, modify the Babel core and therefore call interoperability into question. Similarly, though less dramatically, the MxN effort, initially begun primarily as a visualization tool, has moved toward a more general purpose tool as a high-performance component. More recently the MxN effort has turned toward climate users with the Model Coupling Toolkit as a sort of MxN component specially crafted toward climate applications in general, and the Community Climate Model in particular. These are just samples of a trend in the CCTTSS to use good computer science to enable a more participative community in high-performance computing.

While the CCA and CCTTSS are driven by "what works," serious research in computer science is brought to bear on these problems. Certainly work has become more pragmatic in response to the needs, especially the educational needs, of real users. But simultaneously work has become more esoteric. As CCTTSS investigators stretch their ideas to suit users' problems, serious thinking and rethinking of the basis of their work is required. As a result, the most novel approaches to the solution of these problems have occurred in the last year. There are numerous examples, including Chasm [10], Open Babel [39], and the idea of the CCA design pattern as separate from the specification [24]. Re-inventing how high-performance software is done requires both pragmatic infrastructure and inventive ideas. The CCTTSS is rising to this challenge and is dedicated to remaking the world of high-performance computing from the feudal monarchy of code fiefdoms to a laissez-faire economy of high-performance components.

References

[1] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. C. McInnes, S. Parker, and B. Smolinski, Toward a Common Component Architecture for High-Performance Scientific Computing, in *Proceedings of High Performance Distributed Computing*, pages 115–124, 1999.

[2] Common Component Architecture Forum, `http://www.cca-forum.org`.

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

[4] Common Component Architecture Draft Specification, `http://www.cca-forum.org/specification/`.

[5] Benjamin A. Allan, Robert C. Armstrong, Alicia P. Wolfe, Jaideep Ray, David E. Bernholdt, and James A. Kohl, The CCA Core Specification in a Distributed Memory SPMD Framework, *Concurrency and Computation: Practice and Experience* , 1 (2002).

[6] Madhusudhan Govindaraju, Sriram Krishnan, Kenneth Chiu, Aleksander Slominski, Dennis Gannon, and Randall Bramley, Merging the CCA Component Model with the OGSI Framework, in *Proceedings of CCGrid-2003*, Tokyo, 2003, to appear.

[7] C. R. Johnson, S. G. Parker, and D. M. Weinstein, Component-Based Problem Solving Environments for Large-Scale Scientific Computing, *Concurrency and Computation: Practice and Experience* **14**, 1337 (2002).

[8] Tamara Dahlgren, Thomas Epperly, and Gary Kumfert, *Babel User's Guide*, CASC, Lawrence Livermore National Laboratory, Livermore, CA, 0.8 edition, 2003.

[9] Babel website, `http://www.llnl.gov/CASC/components/`.

[10] Craig Rasmussen and Matthew Sottile, Computer Language Interoperability Using Chasm, seminar, Department of Computer Science, University of New Mexico, Albuquerque, New Mexico, 2003.

[11] U.S. Department of Energy Office of Science, Top 10 Science Achievements in 2002, `http://www.sc.doe.gov/sub/accomplishments/top_10.htm`.

[12] Jim Glimm, David Brown, and Lori Freitag (PIs), Terascale Simulation Tools and Technologies (TSTT) Center, `http://www.tstt-scidac.org`.

[13] Phil Colella (PI), An Algorithmic and Software Framework for Applied Partial Differential Equations (APDEC), `http://davis.lbl.gov/APDEC/`.

[14] David Keyes (PI), Terascale Optimal PDE Simulations (TOPS) Center, `http://tops-scidac.org`.

[15] Rob Armstrong, David Bernholdt, Dennis Gannon, James Kohl, Scott Kohn, Lois Curfman McInnes, Jarek Nieplocha, Steve Parker, and Craig Rasmussen, Center for Component Technology for Terascale Simulation Software, ISIC Proposal to the Office of Science, 2000, `http://www.cca-forum.org/ccttss/overview/ccttss-proposal.pdf`.

[16] Robert Englander, *Developing Java Beans*, O'Reilly, 1997.

[17] Richard Monson-Haefel, *Enterprise JavaBeans*, O'Reilly, 1999.

[18] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG Document, 1998, `http://www.omg.org/corba`.

[19] Object Management Group, *CORBA Components*, OMG TC Document orbos/99-02-05, 1999.

[20] High-performance CORBA Working Group, `http://www.omg.org/homepages/realtime/working_groups/high_performance_corba.html`.

[21] Jon Siegel, OMG Overview: CORBA and the OMG in Enterprise Computing, *Communications of the ACM* **41**, 37 (1998).

[22] Microsoft COM Web Page, `http://www.microsoft.com/com/about.asp`.

[23] R. Sessions, *COM and DCOM: Microsoft's Vision for Distributed Objects*, John Wiley & Sons, 1997, (see also `http://www.microsoft.com/com/about.asp`).

[24] Rob Armstrong, Gary Kumfert, Lois Curfman McInnes, Steve Parker, Ben Allan, Matt Sotille, Thomas Epperly, and Tamara Dahlgren, The CCA Component Model for High-Performance Scientific Computing, paper, Twelfth IEEE International Symposium on High Performance Distributed Computing, Seattle, WA, June 22-24, 2003, submitted.

[25] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, Cambridge, MA, 1996.

[26] G. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA, 1994.

[27] Dennis Gannon, From Web Services to Grid Services, *IEEE Journal on Intelligent Systems* (2003), to appear.

[28] Rachana Ananthakrishnan, Sriram Krishnan, Madhusudhan Govindaraju, Lavanya Ramakrishnan, and Aleksander Slominski, Grid Web Services and Application Factories, in Fox, Berman, and Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 9, Wiley, 2003.

[29] Dennis Gannon, Software Component Technology for High Performance Parallel and Grid Computing, Euro-Par, Springer Verlag, 2001, Invited Keynote Paper.

[30] Dennis Gannon, Randall Bramley, Geoffrey Fox, Shava Smallen, Al Rossi, Rachana Ananthakrishnan, Felipe Bertrand, Ken Chiu, Matt Farrellee, Madhu Govindaraju, Sriram Krishnan, Lavanya Ramakrishnan, Yogesh Simmhan, Alek Slominski, Yu Ma, Caroline Olariu, and Nicolas Rey-Cenvaz, Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications, *Journal of Cluster Computing* **5**, 325 (2002).

[31] Dennis Gannon, Kenneth Chiu, Madhusudhan Govindaraju, and Aleksander Slominski, A Brief Introduction to the Open Grid Services Infrastructure, *special issue on Grid Systems, Journal of Computing and Informatics* (2003), to appear.

[32] S. Krishnan, R. Bramley, M. Govindaraju, R. Indurkar, A. Slominski, D. Gannon, J. Alameda, and D. Alkaire, The XCAT Science Portal, *Journal of Scientific Computing* (2003), to appear.

[33] Lavanya Ramakrishnan, Helen Rehn, Jay Alameda, Rachana Ananthakrishnan, Madhusudhan Govindaraju, Aleksander Slominski, Kay Connelly, Von Welch, Dennis Gannon, Randall Bramley, and Shawn Hampton, An Authorization Framework for a Grid Based Component Architecture, Grid2002 Workshop at SC2002, 2002.

[34] J. D. de St. Germain, A. Morris, S. G. Parker, A. D. Malony, and S. Shende, Integrating Performance Analysis in the Uintah Software Development Cycle, *International Journal of Parallel Processing* (2003), to appear.

[35] J. McCorquodale, J. D. de St. Germain, S. Parker, and C. R. Johnson, The Uintah Parallelism Infrastructure: A Performance Evaluation on the SGI Origin 2000, in *High Performance Computing*, Seattle, WA, 2001.

[36] Gary Kumfert, *Understanding the CCA Standard Through Decaf*, CASC, Lawrence Livermore National Laboratory, Livermore, CA, 2002, DRAFT.

[37] Gary Kumfert, Scott Kohn, Tamara Dahlgren, and Thomas Epperly, Introducing Babel Decaf, Viewgraphs UCRL-PRES-134982, LLNL, 2001, CCA Quarterly Meeting, Bloomington, IN.

[38] Randall Bramley, fortran 90 news, email to cca-forum mailing list, 2002.

[39] Tamara Dahlgren, Thomas Epperly, and Gary Kumfert, OpenBabel Workshop, 2003, half-day meeting.

[40] Common Component Architecture Software, `http://www.cca-forum.org/software.html`.

[41] Alexandria website, `http://www.llnl.gov/CASC/components/alexandria.html`.

[42] High Performance Fortran Forum, *High Performance Fortran Language Specification (Version 1.1)*, 1994, `http://www.crpc.rice.edu/HPFF/hpf1`.

[43] High Performance Fortran Forum, *High Performance Fortran Language Specification (Version 2.0.δ)*, 1996, `http://www.crpc.rice.edu/HPFF/hpf2`.

[44] 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 2.1.5, Argonne National Laboratory, 2003, `http://www.mcs.anl.gov/petsc`.

[45] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith, Efficient Management of Parallelism in Object Oriented Numerical Software Libraries, in E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202, Birkhauser Press, 1997, also available as Argonne preprint ANL/MCS-P634-0197.

[46] LAPACK – Linear Algebra PACKage, `http://www.netlib.org/lapack/`.

[47] The KeLP Programming System, `http://www.cs.ucsd.edu/groups/hpcl/scg/kelp/`.

[48] Daniel Quinlan, A++/P++ - Quick Reference Manual (Version 0.7.5), `http://www.llnl.gov/CASC/Overture/henshaw/documentation/App/Quick_Reference_Manual/Quick_Reference_Manual.html`.

[49] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield, Global Arrays: A Non-Uniform-Memory-Access Programming Model for High-Performance Computers, *J. Supercomputing* **10**, 169 (1996).

[50] Global Array Toolkit Home Page, `http://www.emsl.pnl.gov:2080/docs/global`.

[51] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield, Global Arrays: a Portable "Shared-Memory" Programming Model for Distributed Memory Computers, in *Supercomputing'94*, pages 340–349, Los Alamitos, California, USA, 1994, Institute of Electrical and Electronics Engineers and Association for Computing Machinery, IEEE Computer Society Press.

[52] D. L. Brown, G. S. Chesshire, W. D. Henshaw, and D. J. Quinlan, Overture: An Object Oriented Software System for Solving Partial Differential Equations in Serial and Parallel Environments, in *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, 1997.

[53] H. E. Trease and L. L. Trease, NWGrid: A Multi-Dimensional, Hybrid, Unstructured, Parallel Mesh Generation System, `http://www.emsl.pnl.gov:2080/nwgrid`.

[54] J.-F. Remacle, B. K. Karamete, and M. S. Shephard, Algorithm Oriented Mesh Database, in *Proceedings of the Ninth International Meshing Roundtable*, pages 349–359, 2000.

[55] CUBIT Mesh Generation Environment, Technical Report SAND94-1100, Sandia National Laboratories, 1994.

[56] J. Glimm, J. Grove, X.-L. Li, and D. C. Tan, Robust computational algorithms for dynamic interface tracking in three dimensions, *SIAM J. Sci. Comp* **21** (2000).

[57] L. Freitag, T. Leurent, P. Knupp, and D. Melander, MESQUITE Design: Issues in the Development of a Mesh Quality Improvement Toolkit, in *Proceedings of the Eighth International Conference on Numerical Grid Generation in Computational Field Simulations*, pages 159–168, 2002.

[58] P. Colella et al., Chombo Home Page, `http://seesar.lbl.gov/anag/chombo`.

[59] J. A. Kohl and G. A. Geist, Monitoring and Steering of Large-Scale Distributed Simulations, in *IASTED International Conference on Applied Modeling and Simulation*, Cairns, Queensland, Australia, 1999.

[60] G. A. Geist, J. A. Kohl, and P. M. Papadopoulos, CUMULVS: Providing Fault Tolerance, Visualization and Steering of Parallel Applications, *The International Journal of High Performance Computing Applications* **11**, 224 (1997).

[61] J. A. Kohl, High-Performance Computers: Innovative Assistants to Science, *ORNL Review* **30**, 54 (1997).

[62] P. M. Papadopoulos and J. A. Kohl, A library for Visualization and Steering of Distributed Simulations using PVM and AVS, in *High Performance Computing Symposium*, Montreal, Canada, 1995.

[63] Manish Parashar et al., GrACE Home Page, `http://www.caip.rutgers.edu/~parashar/TASSL/Projects/GrACE`.

[64] Alan Hindmarsh and Radu Serban, User Documentation for CVODES, An ODE Solver with Sensitivity Analysis Capabilities, Technical Report UCRL-MA-148813, Lawrence Livermore National Laboratory, 2002, `http://www.llnl.gov/CASC/sundials`.

[65] MPICH Home Page, `http://www.mcs.anl.gov/mpi/mpich`.

[66] Steve Benson, Lois Curfman McInnes, and Jorge Moré, A Case Study in the Performance and Scalability of Optimization Algorithms, *ACM Transactions on Mathematical Software* **27**, 361 (2001).

[67] Steve Benson, Lois Curfman McInnes, Jorge Moré, and Jason Sarich, TAO Users Manual, Technical Report ANL/MCS-TM-242 - Revision 1.4, Argonne National Laboratory, 2002, `http://www.mcs.anl.gov/tao/`.

[68] High Performance Computational Chemistry Group, *NWChem, A Computational Chemistry Package for Parallel Computers, Version 4.1*, Pacific Northwest National Laboratory, Richland, Washington 99325-0999 USA, 2002, `http://www.emsl.pnl.gov/pub/docs/nwchem/`.

[69] The Massively Parallel Quantum Chemistry Program, `http://aros.ca.sandia.gov/~cljanss/mpqc/`.

[70] C. Janssen, E. Seidl, and M. Colvin, Object-Oriented Implementation of Ab Initio Programs, *ACS Symposium Series 592, Parallel Computers in Computational Chemistry* (1995).

[71] S. Lefantzi, J. Ray, and H. N. Najm, Using the Common Component Architecture to Design High Performance Scientific Simulation Codes, in *Proceedings of the International Parallel and Distributed Processing Symposium*, Nice, France, 2003, accepted.

[72] Sophia Lefantzi and Jaideep Ray, A Component-based Scientific Toolkit for Reacting Flows, in *Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics*, Boston, Mass., 2003, Elsevier Science.

[73] *hypre*: High Performance Preconditioners, `http://www.llnl.gov/CASC/hypre/`.

[74] SuperLU Home Page, `http://www.nersc.gov/~xiaoye/SuperLU`.

[75] Amitava Bhattacharjee (PI), Center for Magnetic Reconnection Studies, `http://www.physics.uiowa.edu/cmrs`.

[76] Steve Jardin (PI), Center for Extended MHD Modeling, `http://w3.pppl.gov/CEMM`.

[77] Paul Hovland, Kate Keahey, Lois Curfman McInnes, Boyana Norris, Lori Freitag Diachin, and Padma Raghavan, A Quality-of-Service Architecture for High-Performance Numerical Components, paper, Workshop on QoS in Component-Based Software Engineering, which is part of the Reliable Software Technologies Conference, Toulouse, France, June 20, 2003, submitted.

[78] P. Beckman, P. Fasel, W. Humphrey, and S. Mniszewski, Efficient Coupling of Parallel Applications using PAWS, in *Proceedings of the 7th IEEE Int'l Symposium on High Performance Distributed Computation*, 1998, `http://www.acl.lanl.gov/paws/`.

[79] SC2001 Conference Activities, Posters and demonstrations in the ANL, LANL, NCSA, ORNL, and Research@Indiana booths., 2001.

[80] The Model Coupling Toolkit, `http://www-unix.mcs.anl.gov/mct/`.

[81] Community Climate System Model, `http://www.cgd.ucar.edu/csm/`.

[82] Tim Killeen, John Marshall, and Arlindo da Silva (PIs), Earth System Modeling Framework, `http://www.esmf.ucar.edu/`.

[83] M. Govindaraju, A. Slominski, V. Chopella, R. Bramley, and D. Gannon, Requirements for Evaluation of RMI Protocols for Scientific Computing, in *Proceedings of SC2000*, 2000.

[84] Filipe Bertrand, Yongquan Yuan, Kenneth Chiu, and Randall Bramley, An Approach to Parallel MxN Communication, in *12th High Performance Distributed Computing (HPDC)*, Seattle, WA, 2003, submitted.

[85] Rajeev Thakur et al., ROMIO Home Page, `http://www.mcs.anl.gov/romio`.

[86] Kenneth Chiu, Madhusudhan Govindaraju, and Dennis Gannon, The Proteus Multiprotocol Library, in *Proceedings of SuperComputing Conference, Baltimore, Maryland*, November 16–22, 2002.

[87] R. J. Duchovic, Y. L. Volobuev, G.C. Lynch, D. G. Truhlar, A. F. Wagner, B. C. Garrett, and J. Corchado, POTLIB 2001: A Potential Energy Surface Library for Chemical Systems, *Computer Physics Communications* **144**, 169 (2002).

[88] POTLIB 2001, `http://users.ipfw.edu/duchovic/potlib2001/`.

[89] ScaLAPACK, `http://www.netlib.org/scalapack`.

[90] 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 (2002), (also available as Argonne preprint ANL/MCS-P932-0202).

[91] Torsten Wilde, James A. Kohl, and Jr. Raymond E. Flanery, A CUMULVS Viewer for AVS/Express, in *2002 Int'l Conference on Computational Science (ICCS 2002)*, Amsterdam, 2002.

[92] Torsten Wilde, James A. Kohl, and Jr. Raymond E. Flanery, CUMULVS Viewers for the ImmersaDesk, in *2001 Int'l Conference on Computational Science (ICCS 2001)*, San Francisco, CA, 2001.

[93] Sameer Shende, Allen D. Malony, Craig Rasmussen, and Matthew Sottile, A Performance Interface for Component-Based Applications, International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'03), 2003, accepted.

A. Component Inventory

This section includes a representative list of components that have been demonstrated at SC2002 and are part of our current RPM distribution (see Sections 2.1.3 and 2.2.2 for further information).

A.1 Utilities and Services in Ccaffeine

Investigators: B. Allan, R. Armstrong, M. Govindaraju, S. Lefantzi, and E. Walsh (SNL)

The CCA specification treats framework services exactly like CCA components except that the port that embodies the service is *always* connected to the component. A number of such services have been added to the Ccaffeine framework during the past year. A parameter port service is the most commonly used and allows components to have parameters set on them interactively by a user. Another service allows the connection of the original Classic ports to Babel components, thereby accommodating legacy CCA software. There are also a variety of utility services that permit a component to access MPI, to receive connection events, and to establish its own interactive window with a user.

A.2 Data Management, Meshing, and Discretization

Global Arrays. Manojkumar Krishnan and Jarek Nieplocha (PNNL). Many scientific applications rely on dense distributed arrays. The Global Array (GA) library [49, 50], provides an extensive set of operations on multidimensional dense distributed arrays. A rather unique capability is the support in GA for the shared memory programming model, where arrays can be accessed as if they were located in shared memory. We developed a `GlobalArray` component that provides interfaces to full capabilities of GA and implements the interfaces for distributed arrays under development the CCA Scientific Data Working Group. This component supports both classic and SIDL interfaces and provides two ports: `GlobalArrayPort` and `DADFPort`. `GlobalArrayPort` offers interfaces for creating and accessing distributed arrays, including linear algebra operations. Some of the linear algebra operations are implemented internally and through interfaces to third-party parallel linear algebra libraries such ScaLAPACK [89]. `DADFPort` offers interfaces for defining and querying array distribution templates and distributed array descriptors following the API proposed the CCA Forum Distributed Data Working Group. The `GlobalArray` component is currently used in applications involving molecular dynamics and molecular shape optimization.

TSTTMeshQuery. Lori Freitag Diachin (SNL, formerly ANL). This classic component is a prototype of an unstructured, triangular mesh component that supports the TSTT mesh query interface [12] for access to node and element geometry and topology information; opaque tags support user-defined data; further information is in [90]. This interface is sufficient to implement linear, finite-element discretization for diffusion PDE operators. The `TSTTMeshQuery` component is currently used in the CCA tutorial to demonstrate the construction of a PDE-based application using CCA components. This component will be expanded to several TSTT-compliant mesh components built from existing DOE software that support a wide range of two and three-dimensional meshes. Such components will be used to demonstrate the utility of interchangeable and interoperable meshing infrastructures in the solution of PDE-based applications.

FEMDiscretization. Lori Freitag Diachin (SNL, formerly ANL). This classic component provides linear, finite-element discretizations for commonly used PDE operators and boundary conditions. It currently

employs unstructured triangular meshes through the `TSTTMeshQuery` component and provides matrix and vector assembly routines to create linear systems of equations in simple PDE-based applications; see [90] for details. The `FEMDiscretization` component approximates advection and diffusion operators as well as Dirichlet and Neumann boundary conditions with either exact or Gaussian quadrature. The component is currently used in the CCA tutorial to demonstrate the construction of a PDE-based application using CCA components. This component uses the `TSTTMeshQuery` and `LinearSolver` ports. This interface is expected to evolve as the TSTT discretization library is developed, and this prototype component will be replaced with a more sophisticated variant that supports multiple discretization schemes and mesh types. Such components will used to demonstrate the utility of interchangeable and interoperable discretization strategies in the solution of PDE-based applications.

GrACEComponent. Jaideep Ray (SNL). This classic component discretizes a domain with a SAMR mesh and implements regriding to preserve resolution and load-balance the mesh. As a wrapper around the GrACE library [63] developed by M. Parashar of Rutgers University, `GrACEComponent` takes care of all the geometric aspects of the mesh (size and location of patches, their resizing due to regriding, and their placement on processes for load-balancing). This component also serves as a factory for a "Data Object" that contains data on all the patches. The data object takes care of message passing for ghost cell updates. This component is used in CFRFS applications and will continue to evolve to incorporate the latest GrACE features.

HODiffusion and SpatialInterpolations. Christopher Kennedy and Jaideep Ray (SNL). These classic components use an underlying Fortran77 library developed by C. Kennedy that implements higher-order (orders 2 – 8) finite difference stencils, including both first and second derivatives. `HODiffusion` supports both symmetric and skewed stencils for collocated and staggered output and calculates higher order diffusion fluxes using these stencils. The `SpatialInterpolations` component supplies the prolongation and restriction operators between SAMR patches at two adjacent levels of refinement, where the order of interpolation has to be commensurate with the spatial discretization in `HODiffusion`. These components currently handle diffusion transport subassembly in CFRFS applications; future plans include continued testing on hierarchical grids.

A.3 Integration, Optimization, and Linear Algebra

CvodesComponent. Radu Serban (LLNL, collaborator in TOPS SciDAC Center). This classic component includes ports both for a generic implicit ODE integrator (`OdeSolverPort`) and for an implicit ODE integrator with sensitivity capabilities (`OdeSolverSPort`). `CvodesComponent` is based on CVODES [64] and is used for chemistry integration in CFRFS applications.

TAOSolver. Steve Benson, Lois McInnes, Boyana Norris, and Jason Sarich (ANL). This SIDL component implements a simple `OptimizationSolver` interface for unconstrained and bound constrained optimization problems; see [90] for details. The underlying optimization solvers are provided by the Toolkit for Advanced Optimization [66, 67] and include Newton-based methods as well as limited-memory variable-metric algorithms that require only an objective value and first order derivative information. `TaoSolver` employs external components for parallel linear algebra, where current support includes both Global Arrays and PETSc. `TaoSolver` is used within applications involving molecular geometry optimization and molecular dynamics, which are further discussed in Sections 2.2.3. `TaoSolver` is the basis for an evolving optimization solver component that will employ linear algebra interfaces under development within the TOPS SciDAC center.

LinearSolver. Boyana Norris (ANL). This classic component provides a prototype port for the solution of linear systems. These interfaces are in the process of evolving to support common interfaces for linear algebra that are under development within the TOPS SciDAC Center. Future work will include transitioning this component, as well as others such as `TaoSolver` and `FEMDiscretization`, to use the new TOPS interfaces, so that they can easily benefit from the full suite of linear algebra software available within the TOPS.

A.4 Parallel Data Description, Redistribution, and Visualization

DistArrayDescriptorFactory. David Bernholdt and Wael Elwasif (ORNL). This classic component provides a uniform means for applications to describe dense multi-dimensional arrays and is based upon emerging interfaces from the CCA Scientific Data Components Working Group, as discussed in Section 2.2.1.a.

CumulvsMxN. Jim Kohl, David Bernholdt and Torsten Wilde (ORNL). This classic component builds on CUMULVS [59, 60] technology to provide an initial implementation of the MxN parallel data redistribution interfaces that are under development by the CCA MxN Working Group. `CumulvsMxN` is designed to span multiple CCA frameworks and to pass data between two distinct parallel component applications. See Section 2.3 for further information.

ParticleCollectionFactory. Jaideep Ray (SNL) and Jim Kohl (ORNL). This classic component is a prototype for doing MxN parallel data redistribution on combustion data for use in post-processing in the CFRFS combustion applications. The `ParticleCollectionFactory` component "fakes" a patch on a SAMR grid as a "particle," which can then be employed for data redistribution and post-processing. Future plans include increasing the robustness of the code and using it in off-machine, concurrent post-processing.

VizProxy Jim Kohl and Torsten Wilde (ORNL). This classic component provides a companion MxN endpoint for extracting parallel data from component-based applications and then passing this data to an external front-end viewer for interactive graphical rendering and exploration. Variants provide general-purpose components for interactive visualization of data based on structured meshes as well as unstructured triangular meshes; support for particle-based data is under development. Using the CUMULVS viewer library and protocols, a variety of commercial and public domain visualization tools can be utilized at the front-end user interface [91, 92]. Currently provided front-ends include a simple 2D "slicer" viewer and a 3D viewer for AVS 5; additional viewers are under development for VTK, AVS/Express and the CAVE. `VizProxy` has been employed in various CCA applications demonstrated at SC2001 and SC2002.

A.5 Graphical Builders and Performance Evaluation

GraphicalBuilder. Boyana Norris (ANL) and Steve Parker (University of Utah). We have developed two complementary prototype graphical builders that can be used to assemble components, set parameters, execute, and monitor component-based simulations. These prototypes have explored different facets of builder functionality: one builder uses XML-based component meta-data to provide a framework-independent component description and the ability to specify a number of query types for component repository searches; the other builder emphasizes compatibility with other component models. These implementations have driven a fine-tuning of the CCA specification to make this interaction more efficient. Graphical builders, also called visual programming interfaces, are very useful for many applications, especially when they are in the prototyping phase. The user interface allows convenient access to component parameters, such as tolerances

and error criteria, which would typically be selected in "what if?" prototyping scenarios and then hard-coded for typical batch runs. As a result, the CCA builder protocol is designed to run with or without the graphical interface. Future work includes working toward a standard way of specifying a user interface to a component. This will allow component writers to specify parameters that they wish to expose to the user, but will not require that they explicitly program user interfaces using a specific toolkit. This approach will simultaneously lower the burden on the component developer, and increase the portability of the resulting system. It is expected that the features of these two systems will merge at some point in the future.

Performance. Sameer Shende and Allen Malony (University of Oregon), Craig Rasmussen and Matt Sottile (LANL), and Jaideep Ray (SNL). The TAU (Tuning and Analysis Utilities) performance observation component provides measurement capabilities to components, thereby aiding in the selection of components and helping to create performance aware intelligent components; see [93] for further details. This component is currently used in CFRFS combustion applications, and future plans include incorporation into a variety of other simulations, including the molecular geometry optimization application, to provide comprehensive inter- and intra-component performance instrumentation, measurement and analysis capabilities.

B. CCTTSS Publications and Presentations

[1] Steve Benson, Manojkumar Krishnan, Lois Curfman McInnes, Jarek Nieplocha, and Jason Sarich, Solving Large-Scale Optimization Problems with Global Arrays and the Toolkit for Advanced Optimization, in preparation for submission to a special issue of Transactions on Mathematical Software (TOMS) on tools within the DOE ACTS Toolkit.

[2] Bill Bosl, Steven Smith, Tamara Dahlgren, Thomas Epperly, Scott Kohn, and Gary Kumfert, Component Technology for Laser Plasma Simulation, paper in preparation.

[3] Paul Hovland, Kate Keahey, Lois Curfman McInnes, Boyana Norris, Lori Freitag Diachin, and Padma Raghavan, A Quality-of-Service Architecture for High-Performance Numerical Components, paper, Workshop on QoS in Component-Based Software Engineering, which is part of the Reliable Software Technologies Conference, Toulouse, France, June 20, 2003, submitted.

[4] Filipe Bertrand, Yongquan Yuan, Kenneth Chiu, and Randall Bramley, An Approach to Parallel MxN Communication, in *12th High Performance Distributed Computing (HPDC)*, Seattle, WA, 2003, submitted.

[5] K. Damevski and S. G. Parker, Parallel Remote Method Invocation and M-by-N Data Redistribution." High-performance Distributed Computing, in *12th High-Performance Distributed Computing Conference (HPDC)*, 2003, submitted.

[6] Rob Armstrong, Gary Kumfert, Lois Curfman McInnes, Steve Parker, Ben Allan, Matt Sotille, Thomas Epperly, and Tamara Dahlgren, The CCA Component Model for High-Performance Scientific Computing, paper, Twelfth IEEE International Symposium on High Performance Distributed Computing, Seattle, WA, June 22-24, 2003, submitted.

[7] S. G. Parker, A Component-based Architecture for Parallel Multi-Physics PDE Simulation, *Elsevier Science* (2003), to appear.

[8] J. D. de St. Germain, A. Morris, S. G. Parker, A. D. Malony, and S. Shende, Integrating Performance Analysis in the Uintah Software Development Cycle, *International Journal of Parallel Processing* (2003), to appear.

[9] J. Ray, C. Kennedy, S. Lefantzi, and H. N. Najm, High-order spatial discretizations and extended stability methods for reacting flows on structured adaptively refined meshes, in *Third Joint Meeting of the U.S. Sections of The Combustion Institute*, Chicago, Illinois, 2003, accepted.

[10] Madhusudhan Govindaraju, Sriram Krishnan, Kenneth Chiu, Aleksander Slominski, Dennis Gannon, and Randall Bramley, Merging the CCA Component Model with the OGSI Framework, in *CCGrid2003, 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan*, May 12–15, 2003, accepted.

[11] Sameer Shende, Allen D. Malony, Craig Rasmussen, and Matthew Sottile, A Performance Interface for Component-Based Applications, International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'03), 2003, accepted.

[12] S. Lefantzi, J. Ray, and H. N. Najm, Using the Common Component Architecture to Design High Performance Scientific Simulation Codes, in *Proceedings of the International Parallel and Distributed Processing Symposium*, Nice, France, 2003, accepted.

[13] S. G. Parker and J. D. de St. Germain, Software Integration in an Academic Environment, DOE Software Quality Forum, 2003.

[14] Craig Rasmussen and Matthew Sottile, Computer Language Interoperability Using Chasm, CompFrame 2003 Workshop, Utrech, Netherlands, 2003.

[15] Rob Armstrong, The Common Component Architecture, in *CompFrame 2003 Workshop*, Univ. of Utrecht, The Netherlands, 2003, `http://www.phys.uu.nl/~steen/Wspage/workshop.html`.

[16] Rob Armstrong, Interoperable Components for Parallel Computing and the CCA, Advanced School for Computing and Imaging, Ninth Annual Conference, 4-6 June, 2003, Heijen, The Netherlands.

[17] Steve Benson, Lois Curfman McInnes, Jorge Moré, and Jason Sarich, The Toolkit for Advanced Optimization, invited presentation in a minisymposium on A Toolkit Approach to Parallel Application Development, SIAM Conference on Computational Science and Engineering, 2003.

[18] Steve Benson, Software Tools for Large-Scale Numerical Applications, invited colloquium, Department of Management Science and Engineering, Stanford University, 2003.

[19] David E. Bernholdt and Gary Kumfert, An Introduction to the Common Component Architecture, invited talk, Joint SWMF-ESMF Interoperability and Model Coupling Workshop, University of Michigan, Ann Arbor, Michigan, 2003.

[20] David E. Bernholdt, Components for Scientific Computing: An Introduction, talk, SIAM Computational Science and Engineering '03, San Diego, California, 2003.

[21] David E. Bernholdt, High-Performance Scientific Software Development: Two Glimpses into the Future, seminar, Quantum Theory Project, University of Florida, Gainesville, Florida, 2003.

[22] David E. Bernholdt, The Common Component Architecture, invited talk, High Productivity Computing Systems Workshop on Software Productivity, Arlington, Virginia, 2003.

[23] Tamara Dahlgren, Thomas Epperly, and Gary Kumfert, Babel: Technical and Cultural Hurdles We've Cleared, and Others Coming In Fast, CASC Works In Progress Seminar Series, Lawrence Livermore National Laboratory, Livermore, California, 2003.

[24] Dennis Gannon, Rachana Ananthakrishnan, Sriram Krishnan, Madhusudhan Govindaraju, Lavanya Ramakrishnan, and Aleksander Slominski, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 9, Grid Web Services and Application Factories, Wiley, February, 2003.

[25] Sophia Lefantzi and Jaideep Ray, A Component-based Scientific Toolkit for Reacting Flows, in *Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics*, Boston, Mass., 2003, Elsevier Science.

[26] Craig Rasmussen and Matthew Sottile, Computer Language Interoperability Using Chasm, seminar, Department of Computer Science, University of New Mexico, Albuquerque, New Mexico, 2003.

[27] Jaideep Ray, Sophia Lefantzi, and Habib N. Najm, CCA-Component Based Simulation of Flows on Adaptively Refined Structured Meshes, in *Minisymposium on Computational Science in Component-Based Environments, SIAM Conference on Computational Science and Engineering*, San Diego, CA, 2003.

[28] Benjamin A. Allan, Robert C. Armstrong, Alicia P. Wolfe, Jaideep Ray, David E. Bernholdt, and James A. Kohl, The CCA Core Specification In A Distributed Memory SPMD Framework, *Concurrency and Computation: Practice and Experience* **14**, 323 (2002).

[29] David E. Bernholdt, A *Brief* Introduction to the Common Component Architecture, talk, Terascale Supernova Initiative Collaboration Meeting, Dallas, Texas, 2002.

[30] David E. Bernholdt, Wael R. Elwasif, James A. Kohl, and Thomas G. W. Epperly, A Component Architecture for High-Performance Computing, in *Proceedings of the Workshop on Performance Optimization via High-Level Languages and Libraries (POHLL-02)*, 2002.

[31] David E. Bernholdt, Wael R. Elwasif, James A. Kohl, and Thomas G. W. Epperly, A Component Architecture for High-Performance Computing, invited talk, Workshop in Performance Optimization via High-Level Languages and Libraries, New York, New York, 2002.

[32] David E. Bernholdt, Wael R. Elwasif, and James A. Kohl, Communication Infrastructure in High-Performance Component-Based Scientific Computing, in Dieter Kranzlmüller, Peter Kacsuk, Jack Dongarra, and Jens Volkert, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 9th European PVM/MPI User's Group Meeting Linz, Austria, September/October 2002. Proceedings*, volume 2474 of *Lecture Notes in Computer Science*, pages 260–270, Springer, 2002.

[33] David E. Bernholdt, Wael R. Elwasif, and James A. Kohl, Communication Infrastructure in High-Performance Component-Based Scientific Computing, talk, 9th EuroPVM/MPI, Linz, Austria, 2002.

[34] David E. Bernholdt, A *Very Brief* Introduction to the Common Component Architecture, invited talk, Earth System Modeling Framework Project Meeting, Greenbelt, Maryland, 2002.

[35] David E. Bernholdt, Component-Based Software for High-Performance Computing: An Introduction to the Common Component Architecture, seminar, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 2002.

[36] David E. Bernholdt, A *Brief* Introduction to the Common Component Architecture, talk, Terascale Supernova Initiative Collaboration Meeting, Dallas, Texas, 2002.

[37] Kenneth Chiu, Madhusudhan Govindaraju, and Dennis Gannon, The Proteus Multiprotocol Library, in *Proceedings of SuperComputing Conference, Baltimore, Maryland*, November 16–22, 2002.

[38] Lori Freitag Diachin, Ben Allan, Rob Armstrong, Steve Benson, David Bernholdt, Jim Kohl, Lois Curfman McInnes, Boyana Norris, and Jaideep Ray, Developing High-Performance Numerical Components, invited presentation in a minisymposium on Software Components for High-Performance Scientific Computing at the SIAM Annual Meeting, Philadelphia, PA, 2002.

[39] L. Freitag, Interface Definition Efforts in the TSTT Center, in *11th International Meshing Roundtable*, Ithaca, NY, 2002.

[40] L. Freitag, Component Technologies for Adaptive Computing, invited keynote, Mississippi State Workshop on Adaptive Algorithms, 2002.

[41] Dennis Gannon et. al., Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications, in *Special Issue on Grid Computing, Journal of Cluster Computing*, volume 5(2002) No. 3, pages 325–336, Kluwer Academic Publishers, July, 2002.

[42] Dennis Gannon, Building Grid Applications from Distributed Component Technology, keynote presentation, PDCS-2002, Louisville, Kentucky, 2002.

[43] Madhusudhan Govindaraju, Sriram Krishnan, Kenneth Chiu, Aleksander Slominski, Dennis Gannon, and Randall Bramley, XCAT 2.0: Design and Implementation of Component based Web Services, Technical Report TR562, Department of Computer Science, Indiana University, Bloomington, June 2002.

[44] C. R. Johnson, S. G. Parker, and D. M. Weinstein, Component-Based Problem Solving Environments for Large-Scale Scientific Computing, *Concurrency and Computation: Practice and Experience* **14**, 1337 (2002).

[45] Gary Kumfert, *Understanding the CCA Standard Through Decaf*, CASC, Lawrence Livermore National Laboratory, Livermore, CA, 2002, DRAFT.

[46] Scott Kohn, Bill Bosl, Tamara Dahlgren, Thomas Epperly, Gary Kumfert, and Steve Smith, Software Component Technology for High-Performance Computing: Babel, CCA, and other Neat Stuff, seminar, San Diego Supercomputer Center, 2002.

[47] Gary Kumfert, Bill Bosl, Tamara Dahlgren, Thomas Epperly, Scott Kohn, and Steve Smith, Babel: Mixing Scripted, Compiled, and Legacy Codes in Adaptive Laser Plasma Physics, talk, SIAM 50th Anniversary and 2002 Annual Meeting. Philadelphia, PA, 2002, Also available as Lawrence Livermore National Laboratory technical report UCRL-PRES-148797.

[48] Lois Curfman McInnes, Tradeoffs in High-Performance Numerical Library Design, invited presentation at the Conference on High Speed Computing, Salishan Lodge, Gleneden Beach, Oregon, 2002.

[49] Lois Curfman McInnes, Steve Benson, Lori Freitag Diachin, Boyana Norris, and J. Sarich, Parallel Numerical Components for High-Performance Scientific Computing, poster presentation at the International Conference on Software Reuse, Austin, TX, 2002.

[50] Lois Curfman McInnes, Steve Benson, and Boyana Norris, Parallel Components for Unconstrained Minimization, brown-bag presentation, University of Chicago - Argonne Computation Institute, 2002.

[51] Lois Curfman McInnes, High-Performance Components for PDEs and Optimization, invited colloquium, Computer Science and Engineering Department, Penn State University, 2002.

[52] J. Nieplocha, R.J. Harrison, M. K. Kumar, B. Palmer, V. Tipparaju, and H. Trease, Combining Distributed and Shared Memory Models: Approach and Evolution of the Global Arrays Toolkit, in *Proceedings of the Workshop on Performance Optimization via High-Level Languages and Libraries (POHLL-02)*, 2002.

[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 (2002), (also available as Argonne preprint ANL/MCS-P932-0202).

[54] S. G. Parker, A Component-based Architecture for Parallel Multi-Physics PDE Simulation, in *International Conference on Computational Science (ICCS2002) Workshop on PDE Software*, 2002.

[55] Lavanya Ramakrishnan, Helen Nell Rehn, Jay Alameda, Rachana Ananthakrishnan, Madhusudhan Govindaraju, Aleksander Slominski, Kay Connelly, Von Welch, Dennis Gannon, Randall Bramley, and Shawn Hampton, An Authorization Framework for a Grid Based Common Component Architecture, in *Proceedings of the 3rd International Workshop on Grid Computing, Baltimore, Maryland*, pages 169–180, Springer Press, November 18, 2002.

[56] Craig Rasmussen, High Performance Components and the CCA, talk, Department of Computer Science, Rice University Houston, Texas, 2002.

[57] Craig Rasmussen and Matthew Sottile, Computer Language Interoperability Using Chasm, seminar, Advanced Computing Laboratory, Los Alamos National Laboratory, Los Alamos, New Mexico, 2002.

[58] David E. Bernholdt, Component-Based Software for High-Performance Computing: An Introduction to the Common Component Architecture, seminar, Pacific Northwest National Laboratory, Richland, Washington, 2001.

[59] David E. Bernholdt, Component-Based Software for High-Performance Computing: An Introduction to the Common Component Architecture, talk, National Center for Atmospheric Research, Boulder, Colorado, 2001.

[60] David E. Bernholdt, Component-Based Software for High-Performance Computing: An Introduction to the Common Component Architecture, seminar, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 2001.

[61] Tamara Dahlgren and Premkumar Devanbu, Components in the Grid, Proceedings of UCD Student Computing Workshop, University of California at Davis, CA, 2001, TR CSE-2001-7.

[62] Tamara Dahlgren and Michael Gertz, The Push and Pull of the Data Grid, Proceedings of UCD Student Computing Workshop, University of California at Davis, CA, 2001, TR CSE-2001-7.

[63] L. Freitag, Component Technologies for High-Performance Computing, invited seminar, Pennsylvania State University, 2001.

[64] Dennis Gannon et al., Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications, invited talk, 2nd International Workshop on Grid Computing (GRID 2001), workshop at SC2001, Denver, Colorado, 2001.

[65] Dennis Gannon, Component Frameworks, keynote address, 10th IEEE High Performance Distributed Computing Conference, San Francisco, CA, 2001.

[66] Dennis Gannon, Comutational Grids, Components and Web Services, Cray Distinguished Lecture Series, University of Minn., 2001.

[67] Dennis Gannon, Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications, keynote address, 2nd International Workshop on Grid Computing, SC2001, Denver, CO, 2001.

[68] Dennis Gannon, Component Architectures for High Performance Grids, talk, The Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC), San Francisco, California, 2001.

[69] Dennis Gannon, Component Architectures for High Performance Grids, invited talk, European Conference on Parallel Computing (EuroPar), Manchester, UK, 2001.

[70] Kate Keahey, Pat Fasel, and Sue Mniszewski, PAWS: Collective Interactions and Data Transfers, talk, The Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC), San Francisco, California, 2001.

[71] Sriram Krishnan, Randall Bramley, Dennis Gannon, Madhusudhan Govindaraju, Rahul Indurkar, Aleksander Slominski, Benjamin Temko, Richard Alkire, Timothy Drews, Eric Webb, and Jay Alameda, The XCAT Science Portal, in *Proceedings of SuperComputing Conference, Denver, Colorado*, November 10–16, 2001.

[72] Gary Kumfert, Bill Bosl, Tammy Dahlgren, Tom Epperlyand Scott Kohn, and Steve Smith, Achieving Language Interoperability with Babel, talk, First CASC/ISCR Workshop on Object-Oriented and Component Technology for Scientific Computing. Wente Vineyards, Livermore, California, 2001, also available as Lawrence Livermore National Laboratory technical report UCRL-PRES-144649.

[73] Lois Curfman McInnes, Component Technology for Terascale Simulation Software, talk, TOPS Center Kickoff Meeting, 2001.

[74] Craig Rasmussen, Component Technology for Terascale Simulation Software, talk, Accelerator SciDAC Center kickoff meeting, 2001.

[75] Matt Sottile, Component Technology for Terascale Simulation Software, talk, LACSI Symposium Workshop on High Performance Numerical Libraries, 2001.

[76] David E. Bernholdt, Component-Based Software Development for High Performance Computing, invited talk, Developing a Toolset for First Principles Electronic Structure Methods Workshop, Oak Ridge, Tennessee, 2001.

[77] David E. Bernholdt, The Common Component Architecture: Status and Plans, invited talk, Accelerated Climate Prediction Initiative Workshop, Oak Ridge, Tennessee, 2001.

[78] Dennis Gannon, Science Portals and Distributed Software Component Architectures, seminar, Department of Computer Science, William and Mary, and ICASE, NASA-Langley, 2001.

[79] Scott Kohn, Gary Kumfert, Jeff Painter, and Cal Ribbens, Divorcing Language Dependencies from a Scientific Software Library, in *10th SIAM Conference on Parallel Processing*, Portsmouth, VA, 2001.

[80] Randall Bramley, Kenneth Chiu, Shridhar Diwan, Dennis Gannon, Madhusudhan Govindaraju, Nirmal Mukhi, Benjamin Temko, and Madhuri Yechuri, A Component Based Services Architecture for Building Distributed Applications, in *Proceedings of Ninth IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, Pennsylvania*, pages 51–59, IEEE Computer Society Press, August 1–4, 2000.

[81] Thomas Epperly, Scott Kohn, and Gary Kumfert, Component Technology for High-Performance Scientific Simulation, in *Working Conference on Software Architectures for Scientific Computing Applications*, Ottowa, Ontario, Canada, 2000, International Federation for Information Processing.

C. Additional CCTTSS Outreach Activities

[1] Craig Rasmussen, Planning for components workshop, Los Alamos Computer Science Institute, to be held at LANL, Los Alamos, New Mexico, April, 2003.

[2] Craig Rasmussen, Working meeting with Ken Kennedy at Rice University to begin collaboration on using the Ccaffeine framework for a telescoping language and optimization system for components, Houston, Texas, 2003.

[3] Lois Curfman McInnes, Working meeting (in San Diego at SIAM CSE Conference) with members of the Terascale Optimal PDE Simulations (TOPS) SciDAC center on prototype common interfaces and component implementations for linear, nonlinear, and optimization solvers, built on top of TOPS software, 2003.

[4] David E. Bernholdt, Computational Science in Component-Based Environments, organized a two-part (8-speaker) mini-symposium (MS21 and MS45) at the SIAM Computational Science and Engineering '03, San Diego, California, 2003.

[5] Craig Rasmussen, Working session with Jay Larson to help develop a joint ESMF and CCA Fortran data coupler component, Los Alamos, New Mexico, 2003.

[6] Tamara Dahlgren, Thomas Epperly, and Gary Kumfert, OpenBabel Workshop, 2003, half-day meeting.

[7] CCA Forum Winter Meeting, Courtyard Los Angeles Old Pasadena, Pasadena, California, 2003.

[8] Rob Armstrong, David Bernholdt, Lori Freitag Diachin, Wael Elwasif, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, and Torsten Wilde, Common Component Architecture Tutorial, CCA Forum Winter Meeting, Pasadena, California, 2003.

[9] Wael Elwasif, Ongoing discussions on the use of CCA in the Psi-Mag toolkit for computational materials science, 2002, 2003.

[10] Tamara Dahlgren, Thomas Epperly, and Gary Kumfert, OpenBabel Workshop, 2002, 1 day meeting.

[11] Craig Rasmussen, Working session with ESMF developers at NCAR regarding CCA and language interoperability, Boulder, Colorado, December, 2002.

[12] Rob Armstrong, David Bernholdt, Lori Freitag Diachin, Wael Elwasif, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, and Torsten Wilde, Common Component Architecture Tutorial, SC2002, Baltimore , Maryland, 2002.

[13] Rob Armstrong and the CCA Working Group, The Common Component Architecture in SciDAC, talk, SciDAC Booth, SC2002, Baltimore, Maryland, 2002.

[14] J. Ray, An Overview of the Common Component Architecture, talk, ASCI Tri-Lab Booth, SC2002, Baltimore, Maryland, 2002.

[15] SC2002 Exhibit Hall presence, Posters, presentations, and demonstrations in the SciDAC, ORNL, ANL, ASCI Tri-Lab, LANL, NASA, PNNL, Research@Indiana, and University of Utah booths., 2002.

[16] Steve Benson, David Bernholdt, Curtis Janssen, Manojkumar Krishnan Liz Jurrus, Lois Curfman McInnes, Jarek Nieplocha, Jason Sarich, and Theresa Windus, Molecular Geometry Optimization Using CCA Components, Demonstrations in the ANL, ORNL, PNNL, and SciDAC booths, 2002.

[17] Ben Allan, M. Govindraju, S. Lefantzi, J. Ray, and D. Gannon, CCA Framework Interoperability, Demonstration at the SCIDAC booth at Supercomputing 2002, 2002.

[18] Rob Armstrong and Tom Epperly, Demonstration of CCA BuilderService: A CCA framework Using Ccaffeine Through Babel-Python, 2002.

[19] Rob Armstrong, David Bernholdt, Lori Freitag Diachin, Wael Elwasif, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, and Torsten Wilde, Common Component Architecture Tutorial, Los Alamos Computer Science Institute Symposium, Santa Fe, New Mexico, 2002.

[20] CCA Forum Fall Meeting, Half Moon Bay Lodge, Half Moon Bay, California, 2002.

[21] Rob Armstrong, David Bernholdt, Lori Freitag Diachin, Wael Elwasif, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, and Torsten Wilde, Common Component Architecture Tutorial, CCA Forum Fall Meeting, Half Moon Bay, California, 2002.

[22] Rob Armstrong, David Bernholdt, Lori Freitag Diachin, Wael Elwasif, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, and Torsten Wilde, Common Component Architecture Tutorial, ACTS Collection Workshop, Lawrence Berkeley National Laboratory, California, 2002.

[23] Craig Rasmussen, Working session with ESMF developers at NCAR regarding CCA and language interoperability, Boulder, Colorado, August, 2002.

[24] Lois Curfman McInnes (organizer), Minisymposium on Software Components for High-Performance Scientific Computing, Invited minisymposium at the SIAM Annual Meeting, Philadelphia, PA, sponsored by the SIAM Activity Group on Supercomputing, 2002.

[25] Craig Rasmussen, Working session with TAU performance group at the University of Oregon to develop a CCA performance monitoring component, Eugene, Oregon, 2002.

[26] CCA Forum Summer Meeting, Argonne National Laboratory, Argonne, Illinois, 2002.

[27] Rob Armstrong, David Bernholdt, Lori Freitag Diachin, Wael Elwasif, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, and Torsten Wilde, Common Component Architecture Tutorial, CCA Forum Summer Meeting, Argonne National Laboratory, Illinois, 2002.

[28] Lois Curfman McInnes, An Introduction to the Common Component Architecture, presentation to visiting students at Argonne National Laboratory, 2002.

[29] CCA Forum Spring Meeting, Valley View Lodge, Townsend, Tennessee, 2002.

[30] Rob Armstrong, David Bernholdt, Lori Freitag Diachin, Wael Elwasif, Dan Katz, Jim Kohl, Gary Kumfert, Lois Curfman McInnes, Boyana Norris, Craig Rasmussen, Jaideep Ray, and Torsten Wilde, Common Component Architecture Tutorial, CCA Forum Spring Meeting, Townsend, Tennessee, 2002.

[31] Don Middleton and David Bernholdt, Discussion of use of CCA in the SciDAC Earth System Grid Project and in NSF ITR proposal "Towards a Knowledge Environment for the Geosciences" lead by Tim Killeen (NCAR), telephone call, 2002.

[32] CCA Forum Winter Meeting, Bishop's Lodge, Santa Fe, New Mexico, 2002.

[33] David E. Bernholdt, Craig Rasmussen, Scott Kohn, and Lori Freitag, Common Component Architecture Tutorial, Santa Fe, New Mexico, 2002.

[34] Lori Freitag, University of Chicago FLASH Center, Bob Rosner PI, Ongoing discussions on design issues in their flagship application code.

[35] Gary Kumfert and Scott Kohn, Introducing Babel: a Language Interoperability Tool, Working Meeting with TOPS (David Keyes, Steve Benson, Andy Cleary, Matt Knepley, Barry Smith) and TSTT (Lori Freitag) representatives, Argonne National Laboratory, Argonne, Illinois, 2001.

[36] SC2001 Conference Activities, Posters and demonstrations in the ANL, LANL, NCSA, ORNL, and Research@Indiana booths., 2001.

[37] Component-Based Software Development in High Performance Computing, "Birds of a Feather" session, SC2001, Denver, Colorado, 2001, Speakers included: Rob Armstrong (SNL), Steve Parker (Utah), Alan Sussman (Maryland).

[38] CCA Forum Fall Meeting, Indiana University, Bloomington, Indiana, 2001.

[39] Lori Freitag, CCTTSS representative to TSTT Unstructured Mesh Query Interface Working Group, 2001.

[40] David E. Bernholdt, Initial discussions with ORNL fusion researchers representing an ORNL-led SciDAC Fusion project and other non-SciDAC projects., 2001.

[41] Scott Kohn, Gary Kumfert, Rob Armstrong, and Dennis Gannon, Workshop on Object-Oriented and Component Technologies for Scientific Computing, 2001, 3 day workshop.