Program Announcements LAB 01-07 and Notice 01-07
ISIC Proposal to the Office of Science
Program Development Environments and Tools

Center for Component Technology for Terascale Simulation Software

Principal Investigator: Rob Armstrong
Sandia National Laboratory
7011 East Avenue
Livermore, CA 94551-0969
Tel: 925-294-2470
Fax: 925-294-1225
Email: rob@sandia.gov

Co-Investigators:

David Bernholdt
Oak Ridge National Laboratory
P.O. Box 2008, Blg. 6012, MS 6367
Oak Ridge, TN 37831-6367
Tel: 865-574-3147
Fax: 865-574-0680
Email: bernholdtde@ornl.gov

Dennis Gannon
Indiana University
Lindley Hall 215
Bloomington, IN 47405
Tel: 812-855-5184
Fax: 812-855-4829
Email: gannon@cs.indiana.edu

James Kohl
Oak Ridge National Laboratory
P.O. Box 2008, Blg. 6012, MS 6367
Oak Ridge, TN 37831-6367
Tel: 865-574-3143
Fax: 865-574-0680
Email: kohlja@ornl.gov

Scott Kohn
Lawrence Livermore National Lab
P.O. Box 808, L-561
Livermore, CA 94551
Tel: 925-422-4022
Fax: 925-293-2993
Email: skohn@llnl.gov

Lois Curfman McInnes
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439-4844
Tel: 630-252-5170
Fax: 630-252-5986
Email: mcinnes@mcs.anl.gov

Jarek Nieplocha
Pacific Northwest National Lab
P.O. Box 999, MS K1-87
Richland, WA 99352
Tel: 509-372-4469
Fax: 509-375-6631
Email: j_nieplocha@pnl.gov

Steve Parker
University of Utah
50 S. Central Campus Dr., Rm. 3490
Salt Lake City, UT 84112
Tel: 801-585-1504
Fax: 801-585-6513
Email: sparker@cs.utah.edu

Craig Rasmussen
Los Alamos National Laboratory
MS B287 CCS-1
Los Alamos, NM 87545
Tel: 505-665-6021
Fax: 505-665-4939
Email: rasmussen@lanl.gov

Executive Summary

Component technology represents an important new tool for software development. Unfortunately, commodity component models that are widely used in industry—such as CORBA, DCOM, and Enterprise JavaBeans—do not address parallelism and other needs of high-performance scientific software. The Common Component Architecture (CCA) component approach specifically targets the needs of large-scale, complex, high-performance, scientific simulations. We have demonstrated the basic principles of such a system. This proposal establishes a distributed Center, comprised of researchers from six DOE laboratories and two universities, focused on taking the CCA from a conceptual prototype to a full-fledged, high-performance component architecture for the scientific community.

We propose a broad research program to define CCA specification standards, create prototype CCA frameworks and related infrastructure, and develop a suite of parallel components to facilitate the initial use of the CCA in other projects. We will also address the increasing need to couple scientific simulations for multi-scale and multi-physics problems by developing tools and techniques for parallel data redistribution among components. In addition, we will take an active role in the integration of CCA technology into scientific applications through work within the Center focused on chemistry and climate modeling and through close collaboration with outside groups adopting the CCA.

This proposal is a logical extension of the software interoperability work begun under the ACTS Toolkit funded by the Office of Science. The resulting component technology will leverage the Office of Science investment in high-performance software and make it available to a wider scientific community.

Table of Contents

Background and Significance

We propose to research, develop, and deploy software component technology for high-performance parallel scientific computing to address problems of complexity, re-use, and interoperability for DOE simulation software. Component technology has revolutionized industry's approach to creating software [Fingarr00,Larsen00,OMG01,Sparling00]. We believe that high-performance component approaches will create a new paradigm for the development of DOE simulation software and numerical libraries. The technology developed under this proposal will provide the following benefits to the DOE computational simulation community:

- Leverage and exploit DOE's legacy software investment by removing barriers to software re-use, enabling collaborative software development rather than individual efforts
- Accelerate software development and reduce costs; computational scientists will develop applications through the integration of existing community components rather than build monolithic applications from scratch
- Enable the integration of high-performance simulation codes with desktop industry software through component software bridges, avoiding the re-creation of existing technology

Component technology represents an important new tool for the development of scientific simulation software. It provides a means to manage the complexity of modern scientific simulation software and enables new simulation capabilities that were previously unavailable due to limitations in library re-use and interoperability. Components allow library developers to describe the calling interface for a software library in a manner that hides low-level details, such as implementation language, compiler, parallelism, or location on a network. Application developers therefore do not need to be concerned with questions such as "Can my Fortran program call this C++ solver library that was parallelized with MPI?" Components encapsulate the knowledge, experience, and work of other scientists, and they provide the building blocks that speed application development.

Component approaches based on CORBA [OMG98,OMG99], COM [Eddon98], Microsoft's dot-NET [NET], and Java technologies [Kara99] are widely used in industry but do not address parallelism. We intend to extend these commodity approaches to satisfy the unique requirements of high-performance parallel scientific computing. Our research activities will address issues of parallel data redistribution among distributed components, integration of high-performance parallel component frameworks with distributed frameworks and existing industry component technologies, and language interoperability with important scientific languages such as Fortran 90. We will develop a single component framework for DOE scientists as well as important components for data management and numerical methods. We will deploy this advanced software technology to the scientific community using a web-based component repository.

This proposal is a logical extension of the software interoperability work begun under the ACTS Toolkit funded by the Office of Science. It has been difficult for applications to use many of DOE's sophisticated software packages due to differences in implementation language, programming style, or calling interfaces. This proposal addresses these issues, and the resulting component technology will leverage the Office of Science investment in scientific software and make it available to a wider scientific community.

Introduction to Component Technology Terms

Component technology is an extension of scripting and object-oriented software development techniques [Ousterhout98, Parker97b] that specifically focuses on the needs of software re-use and interoperability. In simple terms, a *component* is a software entity that adheres to well-defined interoperability behaviors that facilitate re-use [Szyperski97]. Components are software objects (in the object-oriented sense) with additional support for language and network transparency along with required behaviors that simplify communication and connection with other components. In the high-performance parallel computing environment, we define a *parallel component* to be the logical collection of individual but coordinated software components running on multiple processors. This model follows the traditional SPMD programming approach in which a SPMD program consists of multiple but coordinated programs running on a parallel machine. For simplicity, we will typically use the term *component* to refer to both parallel and serial components.

We will use the term *framework* in this proposal to refer to the supporting software infrastructure that enables us to build applications from components. That is, a framework consists of the necessary communication routines, composition support, language interoperability technology, and low-level services required to use component technology. Our use of the term *framework* is similar to the term *Object Request Broker* in CORBA [OMG98]. To use a hardware analogy, a component is like a "software integrated circuit" with well-defined pin-outs that may be connected to compatible pins on other "software integrated circuits." A framework is like the underlying hardware bus that defines standard voltages and bus timing diagrams for communication.

Finally, we will use the phrase *common component architecture* (CCA) to refer to both the proposed overarching software component architecture for DOE high-performance simulations as well as the existing standards body developing that architecture. The term *CCA-compliant* refers to software that follows the CCA architecture specification.

A Motivating Scenario

In this section, we describe a motivating scenario that indicates the potential and broad utility of components in high-performance scientific computing. While this proposal focuses on applications in chemistry and climate modeling, the goals of the Center for Component Technology for Terascale Simulation Software (CCTTSS) are broader and will contribute significantly to other proposed SciDAC work. The following simplified scenario is similar to the established collaboration between CCA researchers and the Computational Facility for Reacting Flow Science, which is submitting a SciDAC proposal in collaboration with members of the CCA (see Appendix C).

In this collaboration, combustion scientists plan to establish a computational facility for parallel distributed-memory chemically reacting flow simulations (see Figure 1). As is commonly the case in this computational domain, their approach uses an explicit time marching algorithm. Implicit methods enable the use of larger time steps and therefore can achieve faster solutions. However, they are not often used in combustion simulations because they require advanced parallel linear algebra capabilities that are beyond the scope of the combustion scientists' primary interests and expertise. In order to introduce an existing linear equation solver component to deal with thin boundary layers, the computational scientists decide to reformulate their existing simulation code as a collection of components. Their principal task in restructuring their application is identifying the interfaces that each component will provide and use, where an interface is a set of function calls that together encapsulate the functionality of the software. In addition, with a small amount of "glue" code to make the overall simulation comply with CCA component standards, other component applications will be able both to use and to be used by the application components.

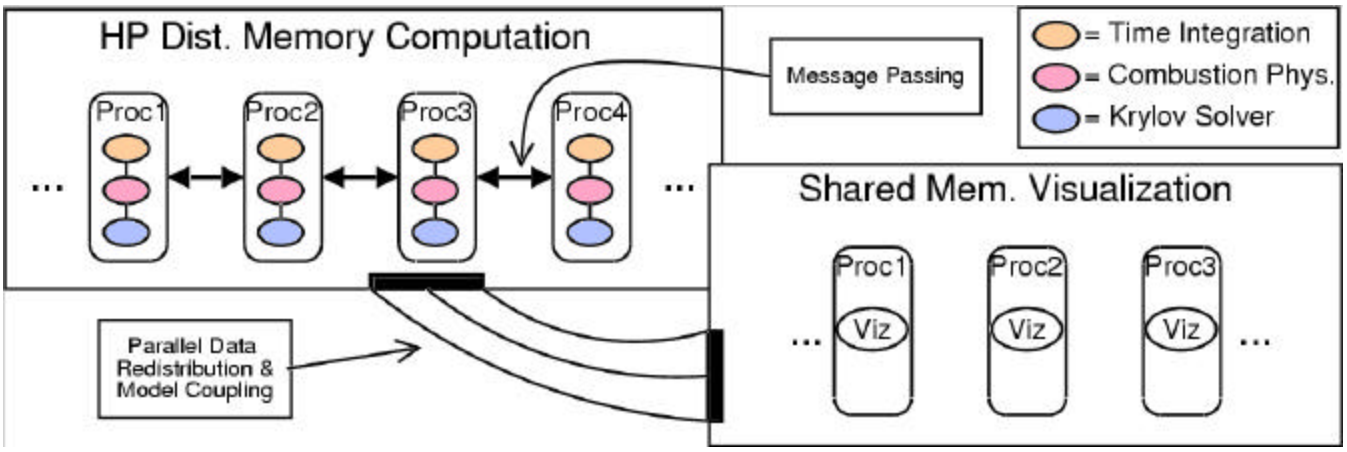

Figure 1: Simplified diagram of a combustion simulation created from components. The computation occurs on a distributed-memory cluster and is composed of three components: a time integrator, a combustion physics model, and a preconditioned Krylov solver. Communication within a parallel component is at the discretion of the component itself. In this figure, each processor communicates with its collaborating processors using MPI message passing. These components encapsulate their respective algorithms, which have been composed to form the simulation (see the framework section for a detailed explanation of composition). Real-time visualization of the simulation takes place on a shared-memory computer, for which the distributed data on the cluster must be mapped by a parallel data redistribution component to the shared-memory machine.

The combustion scientists decide that their application can be split into two components: a multi-scale time integrator and a combustion physics component. In addition, they decide that the linear equation solver will interface with the combustion physics component. They choose to use the reference implementation provided by the Equation Solver Interface (ESI) forum [ESI], whose solvers have been encapsulated as CCA components. The ESI forum involves many DOE numerical linear algebra research groups working together to define common interfaces for parallel linear algebra software. The goal of the ESI is to enable interoperability among different solver toolkits. By using ESI components, the combustion scientists will be able to experiment easily with multiple solution strategies and upgrade to new solver approaches as they become available.

It is likely that the combustion scientists will want to visualize their results on an available shared-memory visualization server. Thus, they will use CCA parallel data redistribution facilities to transfer their processor-decomposed data to a visualization component that encapsulates existing visualization software. Parallel data redistribution is an important research focus area described later in this proposal.

Preliminary Studies and Related Work

In this section, we review previous work by participants of this proposal. We also briefly describe related work by other members of the high-performance software component community. As demonstrated in the following discussion, we have a strong foundation of preliminary research as a multi-institutional collaborative group. Moreover, complementary research projects at our various institutions contribute to our experience base and opportunities for leverage.

The Center for Component Technology for Terascale Simulation Software (CCTTSS) proposal team includes participants from DOE laboratories (ANL, LANL, LLNL, ORNL, PNNL, and SNL) and two academic institutions (Indiana University and the University of Utah). We are a multidisciplinary group with backgrounds in computer science, mathematics, and various scientific application areas. We have a long history of collaboration as members of the Common Component Architecture (CCA) working group. The CCTTSS proposal team is a subset of CCA forum members. The CCA was formed in January of 1998 and is open to all who want to develop component technology standards for high-performance scientific computing. Since its beginning, the members of the CCA have met quarterly to discuss component technologies, jointly develop software, write and vote on component software specifications, write research papers [Armstrong99], and advance the state-of-the-art in component technology for scientific computing. We have also developed a web application called Quorum [Quorum01] to simplify online voting for the CCA. Quorum has enabled the CCA to discuss, vote on, and establish community software standards between our quarterly meetings.

This section discusses preliminary research in four important areas: (1) component frameworks, (2) scientific components, (3) parallel data redistribution tools, and (4) integration of CCA technology into applications.

Component Frameworks

Recall that the term *framework* in this proposal refers to the supporting software infrastructure for component technology. A component framework requires component connection capabilities, a communication infrastructure (for both distributed and efficient same address-space method invocations), language interoperability technology, component composition tools, and a component repository for storing software components. In this sense, component frameworks address different problems than object-oriented application frameworks such as Overture [Brown97], POOMA [Atlas95], and SAMRAI [Hornung98, Hornung01]. Component frameworks focus on the horizontal integration of components across code projects, whereas application frameworks address vertical integration within a narrowly defined application domain.

Proposal participants have a history of investigating the various issues associated with scientific component frameworks. We have implemented both SPMD parallel frameworks and distributed grid-based frameworks. We have experience with communication protocols, performance issues, and language interoperability technology. Preliminary collaborations have resulted in a single CCA component specification (described below) that describes component connection behavior. We will create other community specifications—such as for language interoperability, framework interoperability, and component repositories—under the auspices of this proposal.

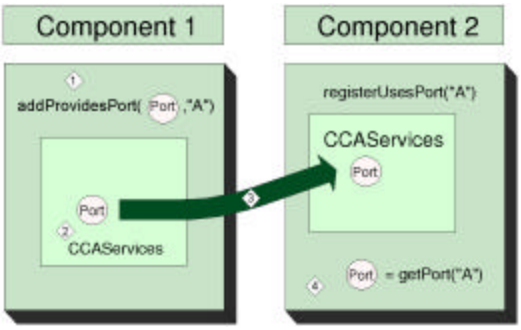

Figure 2: The CCA mechanism for connecting two peer components together using the Provides/Uses design pattern.

CCA Specification

We present a brief overview of the component model and composition mechanisms provided by the current CCA specification [CCA01]. We have focused our effort to date on basic functionality for composing parallel applications from CCA components. The current specification consists of two parts: (1) a core specification defining an interface that a component must implement to be connected to another component (see Figure 2), and (2) a collection of standard interfaces, or *ports*, that components must support. For example, we define standard port interfaces for component connection and disconnection.

We have designed the CCA architecture to be lightweight and simple to use. It is relatively easy to create components from legacy software, thereby bringing interoperability to otherwise monolithic high-performance computing codes. Most importantly, the CCA specification allows the preservation of performance. While the CCA component connection mechanism supports interfaces that are proxies for remote objects, it also supports direct connections between objects in the same memory address space. This approach lowers the latency overhead for using components equivalent to a few virtual function calls, even between languages using the proposed Scientific Interface Definition Language (discussed below) technology for language interoperability. The CCA specification avoids dictating details concerning intra-component communication, thereby preserving latitude to select message-passing, shared memory, or other communication mechanisms that best suit the encapsulated functionality.

Currently the CCA specification is concerned only with composing scientific components. A separate interface definition language called SIDL—for Scientific Interface Definition Language [Cleary98,Epperly00]—is being developed as part of this proposal to provide language interoperability for CCA port interfaces. We plan to express the CCA specification and port interface descriptions entirely in SIDL for language interoperability to C, C++, Fortran77 and 90, Java, Python, MATLAB, and other scientific programming languages, as needed.

Preliminary Framework Research

PARDIS [Keahey97,Keahey97b,Keahey98] is one of the early parallel component framework research activities, and it is one of two academic references cited in the OMG request for proposals for parallel extensions to CORBA. PARDIS, an environment for building PARallel DIStributed applications, employs the important idea of CORBA [OMG98] interoperability through an interface definition language to implement application-level interaction of heterogeneous parallel components in a distributed environment. PARDIS builds on CORBA in that it allows the programmer to construct meta-applications without concern for component location, heterogeneity of component resources, or data translation and marshaling during communication. PARDIS extends the CORBA object model by introducing SPMD objects representing data-parallel computations; these objects are implemented as a collaboration of computing threads capable of directly interacting with PARDIS ORB. The PARDIS approach is related to *PaCo*, which extends CORBA to support efficient encapsulation of parallel codes into distributed objects [René00]. The Ligature project [Keahey00] at LANL focused on providing and processing performance information within a component environment to enable performance-guided design.

The Ligature design approached performance tuning at three levels: application, run-time and hardware. Other work at the Los Alamos Computer Science Institute focuses on compiler approaches for optimizing component software [LACSI].

ORNL has a history of work in message-passing systems and component-based heterogeneous distributed computing environments. The widely used PVM system [PVM] provides a rich operating environment for parallel computing, with message passing, task and resource management, fault notification mechanisms, and pluggable system services. Lessons learned from the PVM research [Geist99] have led to the development of the next-generation Harness environment [Beck99]. Harness is based on the concept of a pluggable virtual machine, where system functionality, messaging substrates, programming models, and the virtual machine itself is "hot swappable" via a parallel plugin mechanism. Plugins dynamically install in parallel in a lightweight kernel. The Harness model is closely related to the CCA model, and Harness will symbiotically feed the design and development of CCA.

Framework for SPMD Computations

The group at SNL has implemented a CCA-compliant SPMD framework called CCAFFEINE [Allan01] that assembles and runs scientific applications from components on commodity clusters. We view this as the starting point for investigating high-performance SPMD components. CCAFFEINE consists of a C++ core framework that allows loading and manipulation of Single Component Multiple Data (SCMD) CCA components and their composition into application codes on workstation clusters. We have implemented the core framework as separable, well-documented pieces, thereby enabling modification or augmentation by the DOE community. CCAFFEINE allows interactive assembly of an application code by a single user. The framework has been created in three independent parts, each of which can be independently reused by applications: a core framework API and implementation, a one-to-many multiplexer, and a GUI. The web browser embeddable GUI visually represents the CCA uses/provides metaphors [Armstrong99]. Aside from the GUI, assembly information can be scripted to the core framework in a native language, thus allowing application codes to be run in a batch queue mode. The multiplexer connects the GUI to the core framework and allows a single user to interact scalably with thousands of processors.

High-Performance Problem Solving Framework

During the past several years, the University of Utah has developed SCIRun [Davidson00,Johnson95,Parker97,Parker99], a scientific programming environment that allows the interactive construction and steering of large-scale scientific computations. SCIRun is a framework in which large-scale simulations can be interactively composed, executed, controlled and tuned. Composing the simulation is accomplished via a visual programming interface to a data-flow network. To execute the program, one specifies parameters with a graphical user interface rather than with a traditional text-based data file. Controlling a simulation involves steering the simulation interactively as it progresses. In SCIRun, the typical components of the computational paradigm—geometric modeling, numerical analysis, and scientific visualization—are integrated into a visual programming environment with the ability to interactively steer any one phase of the process and to see the effects propagate throughout the system automatically. SCIRun currently serves as a testbed for prototyping ideas that will be moved into the CCA specification. Many SCIRun components are already CCA compliant, and as the CCA specification matures, we plan to enhance the framework accordingly. We also intend to use the SCIRun graphical user interface as the basis for a CCA component builder GUI. Finally, SCIRun will be used as an alternative CCA implementation that can validate the CCA claims for true framework-to-framework interoperability.

Framework for Distributed Computations

The group at Indiana University has been involved in two preliminary investigations of software component architectures that are focused on wide-area grid-based environments. Our first efforts in this area are based on a chapter co-authored by Gannon and Grimshaw in the “Grid Book” [Gannon98]. Our first implementation, the "Component Architecture Toolkit" (CAT) [Villacis99], predates CCA and was sponsored by the ACTS Toolkit project. This system explored some preliminary ideas based on a component model where components were linked by connecting ports. This work was demonstrated in several applications, including a component system called the Linear System Analyzer (LSA) [Bramley98,Bramley99], which enabled users to interactively analyze linear systems based on sparse matrix computations. Our first serious application of this system involved integrating LSA into an industrial mold filling simulation [Illinca97] that was part of the HPC Challenge competition at SC'99, where it was awarded the prize for best industrial application. Our second version of the distributed component architecture is based on the original CCA specification [CCA01]. This distributed grid-based implementation was demonstrated at SC'00 and was described in a paper at HPDC [Bramley00].

Language Interoperability

The team at LLNL has developed a tool called Babel [Cleary98,Kohn01,Epperly00] that addresses language interoperability issues for high-performance parallel scientific software. Its purpose is to enable the creation, description, and distribution of language independent software libraries. Babel uses Interface Definition Language (IDL) techniques. An IDL describes the calling interface (but not the implementation) of a particular software library. IDL tools such as Babel use this interface description to generate glue code that allows a software library implemented in one supported language to be called from any other supported language. We have designed a Scientific Interface Definition Language (SIDL) that addresses the unique needs of parallel scientific computing. SIDL supports complex numbers and dynamic multi-dimensional arrays as well as parallel communication directives that are required for parallel distributed components. Babel currently supports Fortran 77, C, C++, and some Python. Babel will provide language interoperability capabilities for our component architecture, and we will add support for additional languages, such as Fortran 90.

Component Repository

The group at LLNL has also implemented a prototype web-based repository called Alexandria to encourage the distribution and reuse of scientific computing components and software libraries [Alexandria01,Epperly00]. Alexandria provides a convenient web-based delivery system for our component technology and thus lowers the barrier to adopting component software. It provides a sophisticated inexact search algorithm, a convenient web-based interface to the Babel language interoperability tool, a repository of type information to be used by the Babel language interoperability tool, and a calling interface to support queries by other component software tools. We will work with the CCA forum to establish common schema for accessing Alexandria from component tools developed by collaborators at other DOE laboratories and academia.

Parallel Components for DOE Computational Science Domains

Proposal team members have developed a variety of prototype components for use in the frameworks described in the previous section. For example, the CCA DataHolder component, which is distributed with the CCAFFEINE framework [Allan01], provides capabilities for parallel management of logically uniform rectangular arrays. In addition, we have encapsulated a rich set of capabilities for parallel numerics, steering, and visualization in software libraries. These libraries provide a starting point for research on domain-specific common component interfaces as well as core technology that will underlie various component implementations. This section discusses preliminary research in community interface standards, parallel numerical toolkit interoperability, and tools for visualization and steering.

Common Data Interfaces

Members of the CCA have recently begun work to define general component interfaces for data associated with arrays and meshes. This work builds on experience encapsulated within various parallel mesh manipulation toolkits, including the DataHolder component provided in the SNL reference framework [Allan01], the SUMAA3d (Scalable, Unstructured Mesh Algorithms and Applications) package [Freitag98], the Opt-MS mesh improvement library [Freitag99,Freitag00], and the GrACE [Parashar00] and SAMRAI [Hornung98,Hornung01] structured adaptive mesh refinement libraries. We also leverage support within CUMULVS [Geist97] for rectangular data arrays, as well as capabilities within the Global Array (GA) system [Nieplocha96], which offers a portable shared-memory programming model for dense multi-dimensional arrays in distributed memory environments.

Another project that provides experience and underlying technology for this work is ARMCI [Nieplocha99], a portable high-performance communication library that supports remote-memory copy and related operations, including I/O vector handling and multi-strided data formats. For performance reasons, ARMCI emphasizes noncontiguous data transfers that correspond to the data structures most often used in scientific applications. We will modify the data interfaces in ARMCI to match those under development for the CCA scientific data interface, in conjunction with ARMCI's support for dense and sparse arrays and stencil representations. In addition, since the GA system is based on ARMCI run-time support and portability, it will be well poised to incorporate parallel data exchange technology, also under development within the Center.

Parallel Numerical Toolkit Interoperability

Several CCA forum participants are members of the DOE Equation Solver Interface (ESI) [ESI] working group, a collaborative effort to develop standards for parallel linear algebra tools within the DOE community. The ESI is committed

to using CCA component technology. CCA forum participants who are also involved with ESI work include the developers of *hypre* [Chow98], ISIS++ [Isis01], PETSc [Balay97,Balay00], and SAMRAI [Hornung98,Hornung01]. We expect that such collaborative efforts in the design of common APIs will facilitate the interoperability among complementary software tools developed at different institutions.

The Advanced Large-scale Integrated Computational Environment (ALICE) project [ALICE01,Freitag99b] has been the center of recent numerical component research at ANL. This project focuses on building low-overhead infrastructure for dynamically assembling parallel software tools and designing performance-sensitive abstract interfaces that define the interactions among them. For example, we developed several light-weight basic services, including error handling, memory management, runtime options management, and component observation [Balay98], which now serve several libraries and provide the foundation for the proposed development of analogous CCA services. ALICE is also exploring interoperability issues among parallel numerical libraries for nonlinear PDEs, optimization, and mesh management, including PETSc [Balay97], TAO [Benson00,Benson00b], and SUMAA3d [Freitag98]. These parallel toolkits use mathematical abstractions in their design, and have been employed in a wide variety of scientific applications. For example, a three-dimensional unstructured aerodynamics application using the PETSc library won a Gordon Bell prize at SC99 [Anderson99]. We have extensive practical experience in developing interfaces among these tools and other software within the ACTS toolkit, including the PVODE ODE solvers [Hindmarsh01], the Overture software [Brown99] for composite meshing and discretization, and several linear algebra packages [Freitag98b,Buschelman00]. The SAMRAI team has also developed interfaces to the Newton-Krylov methods of PETSc, and these tools will be employed in a magnetohydrodynamics application proposal to SciDAC (see Appendix C).

Proposal participants at LLNL have integrated their Babel language interoperability technology into the *hypre* library [Chow98], a suite of parallel scalable linear solvers and preconditioners. This demonstration project enabled the *hypre* team to explore more advanced design alternatives, reduce the size of the library by eliminating redundant routines, and provide calling interfaces for other languages [Smolinski99,Kohn01]. We are continuing this work with the *hypre* team and plan similar collaborations with other members of the SciDAC Terascale Optimal PDE Simulations ETC center (see Appendix C).

Components for Steering, Visualization, and Fault Recovery

The CUMULVS (Collaborative, User Migration, User Library for Visualization and Steering) [Geist97,Papadopoulos95] system developed at ORNL provides a platform for interacting with running simulation programs. It includes functions for run-time visualization of simulation data while they are being calculated [Kohl99]. In response to this visual feedback, scientists can "close the loop" and apply computational steering of parameters on-the-fly, whether model-based or algorithmic [Kohl97]. To maintain the execution of long-running simulations, CUMULVS includes an application-directed checkpointing facility and run-time service for automatic recovery of failed tasks, even in a heterogeneous environment [Kohl98]. The CUMULVS interface uses loose synchronization protocols to extract consistent data frames and apply steering parameter updates in unison across parallel tasks without barriers. Individual steering parameters can be locked to prevent conflicting updates by multiple users. This technology directly applies to the development of visualization and computational steering components for the CCA. The CUMULVS checkpointing and fault recovery system builds on top of the visualization and steering infrastructure. Data descriptions can be marked as contributing to essential program state and collected in checkpoints for failure recovery. A run-time checkpointing daemon handles checkpoints and restores failed tasks automatically. This technology directly applies to fault recovery components and accompanying framework services.

Parallel Data Redistribution and Model Coupling

PAWS (Parallel Application WorkSpace) is a software infrastructure for connecting separate parallel applications within a component-like model [Beckman98]. A central PAWS controller coordinates the linking of serial or parallel applications across a network to allow them to share parallel data structures such as multidimensional arrays. Applications use the PAWS interfaces to indicate which data structures are to be shared and at what points the data is ready to be sent or received. PAWS implements a parallel data descriptor and automatically carries out parallel layout re-mapping when necessary. Connections can be dynamically established and dropped and can use multiple data transfer pathways between applications. PAWS uses MPI and is independent of the application's parallel communication mechanism. An alternative approach for multi-physics model coupling at the level of whole applications is underway at the University of Illinois [de Sturler00].

The CUMULVS [Geist97] data collection interface performs automatic extraction of in-core data from parallel simulations. The application provides data descriptions to specify local data allocation and its context in the global data decomposition. Given this information, CUMULVS transparently extracts desired data elements from a distributed array. This technology will form the basis for parallel data exchange. Complementary to the send/receive data collection in PAWS, CUMULVS protocols support persistent data channels that periodically transmit data frames in sequence. Together, these synchronization protocols relate well to the design of parallel remote method invocations, and CUMULVS and PAWS provide a sound foundation for parallel data redistribution and model coupling.

Integration of CCA into Scientific Simulation Software

To date, CCA participants have focused on developing a component infrastructure and prototype components. However, we have also implemented prototypes of complete component applications. At SC99, SNL and ORNL demonstrated a simple thermal conduction application that integrated an ESI-based numerical component [ESI] and an early version of the CUMULVS-based Eyes visualization component under the CCAFFEINE framework. At SC00, the same team incorporated a combustion model component from another SNL applications group to simulate a chemical reaction-diffusion system. This simulation used swappable implicit solver components and an improved Eyes component in the CCAFFEINE framework [Allan01]. The Indiana University group also demonstrated component technology by integrating linear system analysis capabilities into an industrial mold filling simulation [Illinca97] as part of the HPC Challenge competition at SC'99.

Research Design and Methods

In this section, we describe our proposed research directions that will unite the preliminary work described in the previous section. We present our primary research in four sections. First, we describe the component framework development that will establish a software infrastructure for creating component applications. Next, we list the various types of parallel numerical and data components that we will create for use in scientific applications. The third section discusses probably the most vital research issue facing parallel component technology: parallel data redistribution. The fourth section describes how we will use our component framework and associated components in applications for chemistry and atmospheric simulation. We conclude with a summary of project deliverables and an overview of our software delivery and deployment plans.

Component Frameworks

Coordinator: S. Kohn (LLNL)

The CCA component framework will satisfy three primary goals. First, it will support high-performance parallel applications written in a SPMD style with little, if any, measurable performance overhead. Second, it will support communicating components that execute on distributed parallel platforms (i.e., grid-based programming) at some loss in performance due to network communication costs. Finally, it will interoperate with commodity component architectures such as Enterprise Java Beans or COM or the new Microsoft dot-NET framework [NET]. Application developers will build applications from components, either through a graphical component composition tool, a standard compiled programming language such as C++, or through a scripted language such as Python. We will create a graphical composition tool, and the language interoperability support from Babel will allow developers to connect components in any of its supported programming languages.

Grid Framework Environment

Investigator: D. Gannon (IU)

The most visible application of CCA technology will be the way it enables a coupling between the user's desktop application and remotely executing parallel scientific code. We expect that users will want to compose a parallel component application using components running on different machines. For example, a scientific application may need to connect to a remote database or a remote on-line instrument. In other words, we expect CCA applications will be composed of components that may be executing anywhere on the DOE grid [GF].

We have developed a prototype implementation of the CCA specification that allows the dynamic composition of remotely executing components using Globus [Globus]. This prototype provides a set of services available to any application component. These services include:

- A remote instantiation service, which allows one component to launch instances of other components anywhere on the grid.
- A connection service, exported to components as a port, that can be used to link the provides ports of one component to the uses ports of others. Connections between ports are based on an implementation of Remote Method Invocation (RMI) running over SOAP [Box00]. SOAP is the new XML-based remote procedure call mechanism supported by Microsoft and IBM. Babel XML schemas will be used to define the port interfaces. We recognize that some remote method invocations will require higher performance than SOAP affords. As part of the proposed work, we will create a SOAP-based negotiation mechanism for trading "up" to a faster RMI mechanism.
- A component meta-data archival and search mechanism based on XML specifications that can be stored in a repository such as the Grid Information Service (which is based on LDAP) or on a web based archive such as the Alexandria component repository described below.
- An event service, which allows arbitrary application or framework based event messages to be propagated around the grid to authenticated event listeners such as performance monitors, debuggers, and interactive visualization and steering tools. Each event is an XML object that can be turned into a C++ or Java object automatically.

The services described above are encapsulated as CCA components. This implies that any CCA component can access them in the same way they access any other component. There are several tools available for composing components together into applications. However for large or complicated scientific applications, we use Python scripts. Because Python can also be used as a control language for Microsoft COM, it is not difficult to compose COM and CCA components by building a Python "bridge" component that exposes the interfaces of a COM object as a CCA port. We already have a working, direct link between Microsoft's dot-NET architecture [NET], which is the future of Microsoft technology, and CCA via the SOAP communication.

SCMD Framework Environment

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

Single Program Multiple Data (SPMD) is the most common paradigm for high-performance computing. We have developed a framework called CCAFFEINE that brings component design concepts to SPMD computing. We call this extension SCMD for Single Component Multiple Data [Allan01]. No commercial component model supports a parallel computing paradigm, and none is expected. Figure 3 illustrates our single component multiple data model. Components are linked via CCA ports. Component connections are identical on every processor.

Unlike the distributed grid-based framework that connects components over a network, the connections between components in a SCMD environment are *direct connections*; that is, each interface to the connected component is referenced directly (e.g., through a pointer). The penalty for calling a component through a local interface is a few virtual function calls, at

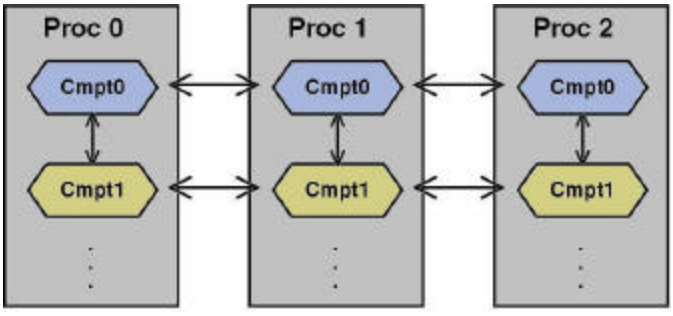

Figure 3: Illustration of intra- and inter-component communication in a SCMD framework. Communication is performed within components (horizontal lines) and direct method invocation is performed between components (vertical lines).

worst. Communication among components on different processors is entirely at the discretion of the components themselves. Although the CCA specification will provide facilities for the use of MPI or PVM, intra-component communication is part of the parallel algorithm and is the component's responsibility.

The proposed work for the first year will focus on the usability of the SCMD framework in the target applications:

- We will embed the SCMD composition infrastructure as a module in Python. This will enable rapid prototyping of SCMD applications for research purposes and reduce the barrier to acceptance by providing scientists with a familiar, well-documented language with which to work. Because scientists can incrementally incorporate CCA-compliant behavior into their existing code as needed, they will not need to buy into everything at once.

- We will develop a proposed standard and implementation for loading CCA SCMD components as dynamically linked libraries. As a part of this work, we will devise a component build standard, and we will produce software tools to facilitate component construction under this standard. Subsequently, we will propose the standard to the CCA forum for adoption. With these tools, a SCMD component written under this build standard will be loadable automatically from any CCA-compliant framework with little porting required by the user.

In future years we will further refine and generalize the SCMD paradigm to accommodate emerging architectures, such as distributed shared memory, and likewise we will augment the CCA standard.

SCMD/Grid Environment Integration

Investigators: B. Allan (SNL), D. Gannon (IU)

This work will focus on integrating commodity components running on desktops to high-performance components running on massively parallel machines. The SCMD framework already contains a multiplexing capability to interface with the distributed-object module. It composes parallel applications on clusters of machines by scripting from a batch file at a single point on the network. As part of the proposed work, we will create a *distributed multiplexer* that will reliably communicate between compositions of SCMD components and the distributed-object, grid-based world, enabling high-performance computing to the desktop. This work will establish a protocol, consistent with the Babel XML schema mentioned below, that will allow users to interact dynamically with components instantiated on tens of thousands of processors.

Concurrency Service Port

Investigator: S. Parker (UU)

We will examine concurrency issues related to a multi-threaded framework. This work will extend previous efforts with multi-threaded programming environments [Parker99]. In particular, we will develop a standard concurrency interface to allow components to use concurrency in a consistent way and an implementation to make it available for community testing. We will also propose modifications to the specification to facilitate thread-based computing on distributed shared-memory architectures. Although parallel computing is typically SPMD, new paradigms are being made available by the advent of distributed clusters of shared memory machines. In the later years of the proposal, we will extend the SCMD concept to include parallel computing paradigms. The goal for the Concurrency Service Interface is to present an intuitive image of concurrency while avoiding its pitfalls. This will require iteration in response to user requirements in subsequent years.

Graphical User Interface to CCA Frameworks

Investigator: S. Parker (UU)

A graphical user interface is an intuitive method to assemble different components into a working application. We will adapt the graphical user interface developed for SCIRun [Parker999] for use in a CCA environment. We will define a standard method for connecting user interfaces to a CCA framework. This will allow other user interfaces to be developed in the future, if such a need arises. We also envision using this interface to allow construction of CCA applications from various scripting languages.

Language Interoperability Technology

Investigators: D. Dahlgren, S. Kohn (LLNL)

Our component architecture will support scientific components written in a variety of common scientific programming languages, such as Fortran 77, Fortran 90, C, C++, Java, Python, MATLAB, and other languages, as needed. Language interoperability is an important capability for a component architecture, as differences in implementation languages frequently hinder the re-use of scientific software libraries.

As described in the *Preliminary Studies* section, we have developed a language interoperability tool called Babel [Epperly00], designed specifically for scientific component software. Babel describes component interfaces using a Scientific Interface Definition Language (SIDL), and Babel uses this SIDL description to automatically generate glue code that mediates calling differences among programming languages. SIDL addresses the unique needs of parallel scientific computing: complex numbers, dynamic multi-dimensional arrays, and parallel communication directives for distributed components.

We will use Babel and SIDL in our component architecture for language interoperability. Because SIDL is language-independent, component tools such as the component repository and graphical application builders will use SIDL type descriptions to describe component interfaces. SIDL type information is stored in an XML format that is amenable to manipulation by automatic component tools. The Babel tools currently support Fortran 77, C, C++, and client-side Python. By the beginning of this proposal, we anticipate that it will also support client-side Java. Under this proposal, we will add language support for Fortran 90 (requested by our atmospheric and astrophysics application collaborators) and MATLAB (for prototyping numerical solvers). Server-side Java and Python will be developed under leveraged funding. We will integrate the Babel technology into the unified component framework and work with the component tool developers to standardize the SIDL XML type schema.

We have also implemented a compiler for the SIDL language that adds support for distributed objects. This system uses Nexus for communication between components located on different machines. We will use this implementation as a testbed for exploring different methods for connecting parallel components that exist on different parallel machines. Successful examples from this testbed implementation will be included into the Babel tool and the CCA specification.

Component Repository and Software Deployment

Investigator: T. Epperly (LLNL)

We will deploy our component technology using a web-based software repository that will house all components developed by this proposal as well as the underlying component framework. We will also work with other ETC centers and application developers to deploy their software using this component repository.

Because component technology opens new opportunities for sharing and re-using software, a component repository must provide sophisticated cataloging, and searching capabilities. The repository must understand component description schemas that describe interface types, versions, and interdependencies. For example, the repository must be able to answer queries such as "Show me all linear algebra solvers that support a matrix of this type."

As described previously, we have developed a prototype web-based repository called Alexandria [Alexandria01,Epperly00]. In addition to providing a web interface for retrieving component software, Alexandria also provides a type repository for SIDL XML interface descriptions as well as a convenient interface to the Babel language interoperability tools. Under this proposal, we will extend Alexandria's support for component software and component tools. We will standardize an XML-based component description schema for storing and retrieving component software. We will define web-based interfaces that enable component tools such as graphical program builders to query and search the repository, obtain component documentation, and automatically download and install software components.

Parallel Components for DOE Computational Science Domains

Coordinator: L.C. McInnes (ANL)

Many difficult research issues must be addressed before we can hope to realize our vision of plug-and-play computational science components. We employ a three-pronged approach: (1) define domain-specific abstract interface specifications through collaborations with other ETCs and applications groups, (2) develop a suite of parallel scientific component software implementations, and (3) explore research issues that are unique to the DOE computational science environment, including quality of service issues related to robust, efficient, and scalable performance. As discussed in the *Preliminary Studies* section, we have a strong foundation for this work in the form of prototype CCA-compliant components and rich parallel tools that already use abstractions in their design.

This ETC will play an essential role in cross-cutting integration activities among other ETCs and application teams by layering and adapting the work of other centers. In particular, we will encapsulate the domain expertise of other scientists by modifying existing parallel software libraries—for example, linear and nonlinear solvers, mesh management, optimization, fault tolerance, steering, and visualization—to use component technology and by collaborating with other ETC teams to develop additional components. This component suite will provide basic functionality needed by a range of simulations, including combustion, fusion, microtomography, 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 abstract 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.

Scientific Data Components

Coordinator: L. Freitag (ANL)
Investigators: B. Allan (SNL), R. Bramley (IU), J. Nieplocha (PNNL), J. Ray (SNL)
Collaborators: proposed Terascale Simulation Tools and Technologies Center (PI: J. Glimm), other proposal participants

The principal underlying data structures in large-scale parallel applications take many forms, ranging from dense and sparse arrays to mesh-based discretizations that approximate continuous PDEs. Because no widely accepted, standardized interfaces exist, applications researchers typically maintain their own unique data structures and interfaces. Linking these one-of-a-kind interfaces with other tools often requires extensive effort, which seriously impedes experimentation with various strategies. We propose to address this situation by developing common interfaces for several data types and deploying them in numerical components. Our interfaces will be flexible enough to support a range of underlying implementations, but specific enough to ensure numerical efficiency and high performance on distributed architectures. The interfaces will support data access at different levels, ranging from low-level raw data formats to higher-level representations, including arrays, meshes, and fields. For the high-level representations, we will pursue three parallel tracks: defining interfaces for multi-dimensional arrays, defining interfaces for mesh-based discretizations, and developing a data interface broker.

Distributed Arrays. For multi-dimensional, distributed arrays, our initial efforts will focus on defining interfaces for dense array formats, including mechanisms for accessing data by subblocks or in a strided fashion. We will also define interfaces for several sparse storage formats and consider additional operations such as gathers/scatters for dense and sparse arrays. We will evaluate interface efficiency and flexibility in tests with existing distributed array implementations, including Global Arrays [Nieplocha96], which serves the NWChem [NWChem01] application discussed in the *Application Integration* section.

Mesh-based Discretization. For mesh-based discretization, we will initially focus on defining interfaces to access static, non-adaptive data on both structured and unstructured meshes. Once the initial specifications are complete, we will work to define higher-level interfaces that support common global operations on meshes, fields, and particles and that support mesh modifications such as adaptive refinement. We also will work to support advanced functionality needed for hybrid meshes and discretizations. We will evaluate interface flexibility and efficiency by creating component implementations of existing

mesh management packages and testing these implementations in applications, such as the complex chemistry flame simulations proposed in [Najm01]. As particular examples, we will create CCA components using the GrACE [Parashar00] structured adaptive mesh refinement library and the SUMAA3d [Freitag98] unstructured adaptive mesh refinement codes. We also will develop interfaces for interacting with load-balancing, solver, visualization, and I/O components.

Data Interface Broker. Because no single interface efficiently supports all data types, we will develop a Data Interface Broker (DIB) to avoid redundant data transformations and to ease component interconnections. This memory-resident, type-safe interface broker component will manage alternative interfaces (or views) of the same data. This capability, in turn, will allow the application code to be formulated as a stream of algorithmic components that each transform the named data according to its desired view. To test DIB functionality, we will develop a prototypical transformation component that can extract a named data interface from the broker, create an alternate interface to its data, and contribute the new interface back to the broker. In general, these transformation components would be contributed by domain experts and, initially, be manually applied by the application composer. Because some transformations may entail significant data access overheads, we will explore mechanisms to estimate and publish the one-time and recurring transformation costs.

To ensure the appropriateness of our data interfaces, we will work closely with developers of mesh management software, such as those within the proposed TSTT Center (PI: J. Glimm). We will also work with developers of complementary components for data redistribution and solvers. We will broaden our outreach activities to include several ETC teams, in particular those involved with solvers and visualization. Finally, to ensure wide adoption of the technology, we will solicit participation from the scientific community through interface specification review, tutorials, and workshops.

Components for Nonlinear Solvers and Optimization

Investigators: L.C. McInnes (ANL), S. Benson (ANL) and B. Norris (ANL)
Collaborators: Terascale Optimal PDE Simulations Center (PI: D. Keyes), Center for Scalable Implicit Nonlinear Extended Magnetohydrodynamics (PI: J. Finn).

The scalable solution of nonlinear PDE and optimization problems pervades many DOE computational science applications, including fusion [Finn01,Jardin01] and chemistry [Janssen95,NWChem01], and it poses difficult challenges in terms of interfaces, algorithms, and implementations. We propose to explore numerical component design for linear and non-linear solvers and related optimization software, as well as low-level service components for managing runtime options, error handling, profiling, and tracing.

Optimization Software. We will develop parallel optimization components, with initial emphasis on unconstrained and bound constrained minimization. The Toolkit for Advanced Optimization (TAO) [Benson00,Benson00b] will provide the foundation for this work. While this software has broad applicability in many scientific disciplines, research in this proposal focuses on collaborations with computational chemists to design interfaces between TAO and the applications MPQC [Janssen95,Janssen96] and NWChem [NWChem01]. Our goal is to encapsulate the complementary expertise of mathematicians and chemists in well-defined components, so that together, we can address terascale electronic structures computations. Since TAO, MPQC, and NWChem have all been designed from their inception using abstract interfaces, we have a solid starting point for this work. We will initially focus on unconstrained minimization problems using Newton-type algorithms, where natural interface points are at the levels of function (or, in this case, energy), gradient, and Hessian evaluations, as well as application-specific convergence monitoring. See the *Application Integration* section for additional details. We will use the interfaces under development by the ESI forum [ESI] for the linear subproblems, thereby enabling the incorporation of new advances in parallel linear algebra that may become available in ESI-compliant toolkits under development throughout the community. After completing this initial phase of work, we will consider problems with nonlinear constraints and will investigate new optimization strategies that exploit application-specific knowledge.

Nonlinear Solvers and Derivative Component Factories. We will develop parallel nonlinear solver components, with emphasis on implicit solution strategies using Newton-Krylov techniques within the PETSc [Balay97,Balay00] library. Such methods have proven very effective within large-scale PDE-based problems through synergistically combining the fast convergence of Newton-type methods for nonlinear systems with preconditioned Krylov techniques for linear subproblems [Anderson99,Gropp00,Mousseau00,Pernice01]. We will focus initially on component interfaces at the level of function and Jacobian evaluation, and we will use ESI interfaces for the linear subproblems. In addition, we will develop component

factories [Gamma95] capable of generating derivative components for Jacobians (and the related components for applying these linear operators to vectors for use in matrix-free methods). These component factories will compute derivatives via finite differences and automatic differentiation [Griewank00] and will build on preliminary work [Abate00] in interfacing between the automatic differentiation tools ADIFOR [Bischof96] and ADIC [Bischof97] and the toolkits PETSc and TAO. We will also examine the use of component factories for slicing, chopping, and other types of software refactoring. For example, a component for computing a nonlinear function (the "input" to the component factory) could be used to generate a Jacobian component via automatic differentiation, and unneeded auxiliary computations within the function component could be removed via software chopping. We will evaluate the performance of these tools with a variety of scientific applications.

Abstract Interface Definition. We will collaborate with ETC and applications groups to develop suites of common abstract interfaces for nonlinear PDEs and optimization software. In particular, we will work with members of the proposed Terascale Optimal PDE Simulations Center (PI: D. Keyes) and the proposed Center for Scalable, Implicit, Nonlinear, Extended Magnetohydrodynamics (PI: J. Finn) to develop suites of abstract interfaces for parallel Newton-Krylov methods. Our goal is to enable plug-and-play functionality for easy experimentation with new algorithmic capabilities, including techniques for derivative computations within Jacobian and Hessian matrices, line search and trust region methods, and solvers for linear subproblems. The involvement of both algorithmicists and applications specialists in interface definition is critical, as we aim to define interfaces with sufficient generality to support a range of scalable implementation strategies.

Quality of Service Research. 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 quality of service (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. We will also investigate how QoS specification mechanisms can take advantage of tools for runtime performance prediction and adaptation, such as Active Harmony [Hollingsworth98] and Autopilot [Ribler99].

Computational Steering and Interactive Visualization Components

Investigator: J. Kohl (ORNL)

A fundamental need in many scientific models is the ability to analyze the ongoing behavior of the simulation and to maintain its valid progress. An intuitive approach to this analysis is via interactive visualization. Data fields being computed or operated on by simulation components can be collected and passed to front-end visualization components that render and graphically display the data. This viewing can reveal the current computational state or animate the progress of a simulation. We will apply existing visualization technology, such as that provided by CUMULVS [Geist97] and PAWS [Beckman98], to develop a general-purpose visualization component for CCA. The resulting component will interface to a variety of commercial and public domain visualization tools. Because simulation components can be parallel, it will be necessary to extract parallel datasets from various distributed decompositions. This capability will be provided by CCA parallel data redistribution technology (discussed in the following section), which will pull the desired data elements to provide a viewable data subregion, whether at a serial or parallel front-end viewer. ORNL is also working with SNL and LANL to design a specialized service for interacting with the underlying message-passing system in a CCA framework. Such capabilities are useful for high-performance out-of-band communication among CCA components. The framework service will provide component grouping information that would otherwise be difficult to determine without framework assistance.

While monitoring a simulation using interactive visualization, a scientist may often want to steer or change the course of the simulation while it is running. These changes could take the form of updates to the values of predefined steering parameters, adjustments of algorithmic or numerical features, or perturbations of physical parameters in the scientific model. Alternatively, the scientist may decide to truncate the execution of malformed simulations or experiments that have gone awry. We will apply the computational steering interface and protocols in CUMULVS to create a CCA steering component. The CUMULVS protocols scatter and synchronize steering requests among all cooperating tasks in a parallel component to guarantee that steering updates are applied in unison. Locking mechanisms also ensure that conflicting updates are not made

simultaneously to a given steering parameter. Front-end methods will be provided by the steering component to enable interactive steering from a GUI, as well as automatic steering by software-based controller components.

Fault Tolerance for Parallel and Distributed Computation

Investigator: J. Kohl (ORNL)

Experience with today's parallel architectures indicates that faults can occur with a frequency sometimes shorter than that of a typical simulation run. We will develop fault recovery mechanisms for CCA that allow uninterrupted execution of extended high-performance simulation runs. We will encapsulate existing facilities for fault tolerance, such as those provided by CUMULVS [Geist97], to develop components that implement a variety of checkpointing schemes. These components will also oversee recovery of component data after a failure. We will implement other aspects of fault recovery within an underlying framework service. Fault notification events will be added via a fault monitor service, to provide detection of faults and failures. As part of this service, callbacks will be provided to notify components of failures and to allow automatic recovery handling or adapting around the faults. The service will monitor both computational and network resources within the framework, as well as catch software or numerical failures within components.

Multi-Threading and Load Redistribution Components

Investigators: C. Janssen, H. Adalsteinsson, and M. Leininger (SNL)

Multi-threading can be a useful way to overlap communication and computation and thus hide communication latencies in parallel applications [Nielsen00]. This approach is used in the Massively Parallel Quantum Chemistry package (MPQC) [Janssen95], which is being developed into components, as described in the *Application Integration* section of this proposal. As a part of this work, we will develop two general components that provide multi-threaded capabilities: one that represents threads and allows users to overlap computations and communication, and one that uses multi-threading to distribute work asynchronously to parallel applications and thereby to reduce load imbalances.

Parallel Data Redistribution and Model Coupling

Coordinator: J. Kohl (ORNL)
Investigators: D.Bernholdt (ORNL), K.Keahey and C.Rasmussen (LANL), S.Kohn and G.Kumfert (LLNL), S.Parker (UU)
Collaborators: Other proposal participants

A core problem faced by high-performance scientific simulations is the coordination of distributed data among parallel tasks. Data are often divided into subsets and scattered or copied across parallel components to improve communication patterns and locality of access by remote tasks. These data decompositions must be decoded and mapped for data sharing among coupled parallel components and for extracting data elements for visualization or checkpointing. Because each parallel component can distribute data on different numbers of processors in unique formats, such data collection can require complex redistribution operations. Tools for parallel data redistribution and extraction are core technologies for reducing the complexity of model coupling, visualization, and other parallel data sharing. Our goal is to enable the efficient composition of distinct parallel simulations as well as safe exchanges of parallel data among them. The set of potential data operations can be divided into several transformational "filters" that perform the relocation of data elements, unit conversions, temporal and spatial interpolations, and translations among disparate meshes.

Another related but distinct issue with cooperating parallel components is their ability to invoke methods on each other. Functions could be called to perform parallel operations or update state in parallel, possibly returning computed results in serial or parallel arrangements. This capability is referred to as Parallel Remote Method Invocation (PRMI). Supporting PRMI is a problem unique to the CCA, as currently no commercial systems provide such functionality. The CCA programming model requires semantics, policies, and conventions for invoking parallel methods and appropriately communicating function arguments and results. Synchronization is also a fundamental concern to insure consistent invocation ordering and parallel data communication, avoid deadlocks, and handle various failure modes.

MxN Parallel Data Exchange Using a Two-Pronged Approach

Developing general infrastructure for parallel data exchange, which lies at the heart of both model coupling and PRMI, is a daunting task. The efficiency and utility of solutions depend on data representations, phases of computation, interaction and access patterns, and the underlying framework implementation. The flexible and dynamic nature of component-based frameworks exacerbates these challenges. Data communication among parallel components requires more general support than that of a typical, monolithic, parallel SPMD code. Because scientists can connect pre-existing components at runtime, component developers cannot know all possible communication patterns *a priori*. For example, a physics component may distribute its data across M processors of a supercomputer, while at run time a user may connect this physics component to a parallel visualization component that runs on N processors of a graphics computer, where M and N are different. A fundamental research challenge is the development of general infrastructure that supports efficient "MxN" communication and parallel redistribution of data between components.

We will develop a specification and tools for redistributing a variety of parallel scientific data structures. The tools will operate on information provided by application components via scientific data interfaces or by instances of the data interface broker (discussed in the previous section). A number of research projects have addressed parallel data redistribution for the special case of rectilinear arrays or other specialized structures. CUMULVS [Geist97] and PAWS [Beckman98] support redistribution among different numbers of processors for arrays and unstructured particle data. Additional work on PARDIS [Keahey97,Keahey97b,Keahey98] demonstrated that parallel communication streams between sets of processors can significantly reduce the cost of data redistribution for distributed arrays. However, general solutions for arbitrary data structures do not yet exist, and thus parallel data redistribution remains an unsolved challenge. The scientific data interface effort aims to alleviate this burden by providing specifications and access to data with arbitrary distributions. Our MxN solutions will capitalize on these interfaces to enable parallel data exchange for a variety of meshes and data structures. We will explore the ARMCI [Nieplocha99] library as a messaging substrate on which to build the actual data movement.

We will consider two distinct, complementary approaches to the interface and implementation of MxN parallel data redistribution technology. Each approach focuses on specific goals in terms of flexibility and user-friendliness. We will pursue *both* approaches to carefully balance interface simplicity with its extensibility and thereby to address the needs of both casual and advanced users.

In the first approach, we will investigate high-level data redistribution components that reside at the application level, above the base framework. This approach involves the direct invocation of component methods for each operation and thereby has considerable flexibility and precise semantics for exerting *explicit* control over redistribution functions. This approach is extensible because additional components can be independently introduced without modification of the framework or base interface specification. However, the user must assume responsibility for understanding and invoking all data redistribution methods from within application components. This approach favors more sophisticated users over non-expert users.

In the second approach, we will investigate core modifications to component interface descriptions and additional low-level framework services that support *implicit* parallel data redistribution. This approach hides many of the details from the user, but increases cost in terms of framework complexity and some possible loss of generality. Under this approach, 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. Each framework instance or component implementation will include configuration parameters and settings to control this automatic handling. In this approach, less sophisticated users need not explicitly translate parallel data references when making method invocations. However, more burden is placed on component and framework developers to supply the necessary redistribution implementations.

Ultimately, the two approaches can be combined given some enhancements to the basic CCA framework services model. Implicit redistribution services could share explicit component-based solutions if framework services are made pluggable. Similarly, individual method implementations for redistribution could be replaced with proxies to redistribution components that would provide the desired functionality. We will evaluate the advantages, disadvantages, and feasibility of each approach. The following subsections describe these two approaches in more detail.

MxN Component Solution

One solution for parallel data redistribution and model coupling involves the development of distinct, specialized components that exchange and translate data. The core problem when sharing data between parallel components is the mapping and communication of the actual data elements. Each parallel component uses its own data decomposition, according to its performance needs and access patterns. The MxN component will understand these different decompositions and determine which data elements should be communicated. The MxN component will be instantiated into the CCA framework like any other user-level component. All components wishing to utilize MxN functionality will connect to appropriate ports on the MxN component. Any attached components will be permitted to share and exchange data with each other. The apparent bottleneck of routing massive amounts of parallel data through a single MxN component is readily alleviated by implementing the MxN component itself in parallel, as shown in Figure 4. Each thread of a parallel component will connect to its own thread of the parallel MxN component, which will perform all necessary data communication internally within its own parallel substructure.

Interfaces for MxN parallel data exchange will describe the data organization in each given parallel component. This includes a specification of the local data layout and allocation for each thread or process, as correlated to the global data decomposition to provide a context for the local elements in the effective overall array. Once this information is known, an interface is needed to engage the parallel data transfer. We will provide methods to register a data field so that it is available for parallel exchange, to indicate when a given local data field is ready for transfer, and to request the actual exchange.

Existing technology for parallel data redistribution employs several synchronization modes. We will generalize these alternatives and provide a uniform specification that encapsulates them. One mode uses simple send/receive semantics to coordinate "one-shot" transmissions of parallel data from source to destination. This explicit approach, such as provided by PAWS, allows arbitrary invocations of parallel data redistribution throughout execution. In this case, although the parallel communication schedule can be saved, the actual data channel and inter-component synchronization are not maintained between invocations. Another mode creates a persistent data channel for periodically sending copies of parallel data between the source and destination. This approach, based on CUMULVS, sets up a regular interval for transmitting "frames" of parallel data and maintains a loose synchronization among components. This approach is particularly useful for animation using a steady sequence of data snapshots, and for model coupling where two iterative components periodically reconcile data. Each parallel thread automatically sends its next data snapshot at the predetermined iteration count, or "frequency".

Exchanging elements from parallel, 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 translations beyond simple MxN data mapping could be needed. If the source and destination data use different meshes or spatial coordinate

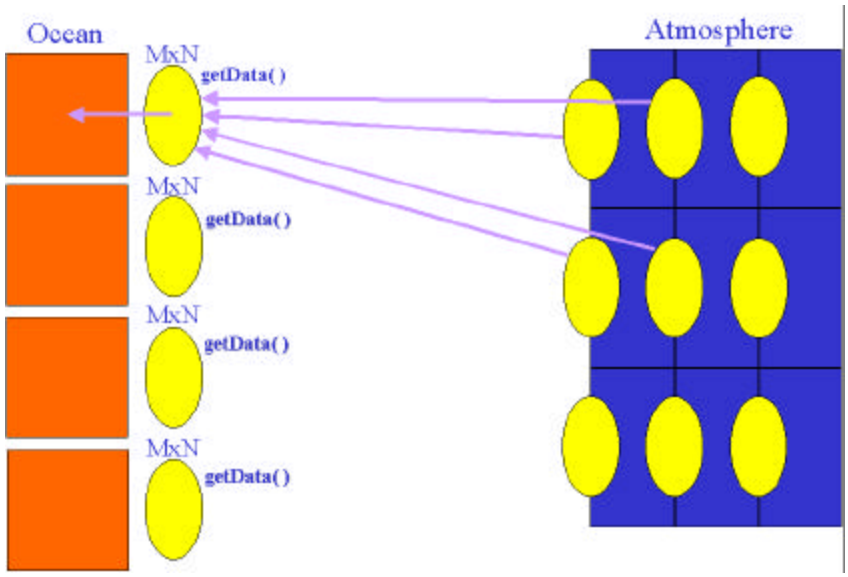

Figure 4: Parallel MxN component model coupling an ocean code with an atmospheric code.

representations, or are computed in different units or at different time steps, then a pipeline of several translation and conversion components will be needed to transform and share semantically comparable parallel data. A wealth of interpolation and sampling schemes are available, although historically, such schemes carry with them an almost religious stigma, and there is much debate among scientists on the merits of one approach over another. We will define appropriate component interfaces with hooks to support generic data transformations and conversions. Given sufficient interface flexibility, various implementations can be developed to cover a range of interpolation and conversion algorithms. Guided by the specific needs of chemistry and climate applications, among others, we will develop generic conversion interfaces and create several components to handle simple data translation schemes.

An important pragmatic issue is how well redistribution functions "compose" with each other. Efficient methods are needed to apply each data translation by components concatenated in a redistribution pipeline. We will explore techniques to operate on data "in place" and to avoid unnecessary data copies. We will explore "super-component" solutions by combining several successive redistribution and translation components into a single optimized component. Such work has already been explored for numerical libraries [GuL99], and approaches based on scheduling can also improve performance [Keahy00].

MxN Framework Service Solution

Our second approach attempts to hide the application developer or component user from details of remote parallel components and data redistribution issues. In order to simplify software development for the applications developer, this approach places more responsibility upon component developers and the underlying CCA framework. The framework will be responsible for data communication between parallel computers, and components will be responsible for describing their data layout to the framework.

The component developer will typically either be hoisting legacy code into a component or creating new software outright. The developer will build components by creating interface descriptions and then implementing these interfaces. To handle parallel data redistribution automatically, additional interfaces must be defined and implemented within the component. We will provide several such redistribution interfaces, each implementing a specific type of data redistribution and varying in its degree of flexibility and performance. The developer will choose from among these interfaces and implement the required methods based on the component's parallel data input and output needs.

To support these interfaces at the framework level, we will develop a generic multiplexor (MUX) service. This MUX will be responsible for maintaining intimate knowledge of all redistribution interfaces available for each component, and will perform the appropriate transformations. The MUX will generate routing tables and schedules for data redistribution, and will handle remote parallel data holder creation and destruction. The MUX will verify parallel data arguments against the user's data layout at method invocation time, and will convert method arguments to match the target data layout before method execution. The MUX will convert any invocation results back into the original data layout before the method completes. Because component users will create the actual references to parallel components, any data layout details will be implicit in the given user code, and thus will not be directly specified to the MUX. The users will write the code that utilizes "redistribution friendly" components, and ultimately will be responsible for reasonable performance decisions.

Parallel Remote Method Invocation

An interface for Parallel Remote Method Invocation (PRMI) is essential for any framework that supports interactions among parallel components. A policy is required to define expected behavior for parallel invocations and when handling partial execution or failure. Interfaces must accept both scalar and parallel data for method arguments, as well as for return values. Appropriate semantics are needed to interpret such uses, such as whether scalar method arguments are copied to every cooperating thread or sent only to a designated one. The corresponding implementation will be closely tied to the available mechanisms for data redistribution and translation. Coordinating the synchronization of invocations is also important to avoid potential pitfalls such as deadlock.

Our approach to PRMI will have the "look and feel" of standard industry frameworks such as CORBA. However, we will apply our data redistribution technology to support parallel data. We will add grammatical constructs to SIDL to assist with the specification and interfacing to PRMI. For example, we will add a method modifier "local" to indicate whether that a method is only meaningful for local references within a parallel component and not for remote component references. We are also interested in developing a high-performance encoding that is interoperable with industry systems. SOAP [Box00] is

an emerging industry standard for encoding interoperable remote method invocations. The SOAP standard defines how to encode remote method invocations using an XML document and then transport them across a network via generic protocols such as HTTP or SMTP. Unfortunately, SOAP does not efficiently encode large blocks of raw data [Govindaraju00]. We will investigate extensions to SOAP that add optimizations for efficient large-scale data transport. We will utilize simple software bridges to connect our frameworks to industry frameworks as they become available. We will also explore the use of MIME encodings, which are de facto standards for transferring binary information.

Integration of CCA into Scientific Simulation Software

Coordinator: D. Bernholdt (ORNL)

Since the Common Component Architecture is an enabling technology for a broad spectrum of software development activities, we will take an active role in promoting and supporting the integration of CCA technology into user software following a three-part strategy. Part one involves general outreach to prospective users to increase awareness of the CCA and its benefits. All Center members will participate to varying degrees in this effort through papers, talks, tutorials, demonstrations, etc. The diversity of experience of Center members will facilitate reaching a broad community.

We expect many projects to integrate CCA technology into their research programs. We are already aware of numerous projects planning to use the Center's work, as described in Appendix C. However, CCA software and tools will not be "thrown over the fence" to these researchers. As the second part of our strategy, Center researchers will maintain close contact with outside adopters, acting both to support the adopters and to insure that the Center receives feedback. Again, all Center members will participate to varying degrees, based primarily on expertise and location. Adopters may also make more extensive use of the ETC by providing additional direct support to Center researchers; such arrangements are anticipated by several projects.

These activities will be complemented by a focus within the Center on the integration of CCA technologies into applications in climate modeling and chemistry. These are domains of particular relevance to DOE and SciDAC. There are significant opportunities for CCA that would likely be missed without this focused effort. By performing this work primarily within the Center (though still in close collaboration with domain specialists), we will also provide an environment in which to examine CCA research questions, a tighter loop for feedback on the usability and performance of Center products, and a testbed for new tools before they are released to the larger community. This work will also provide substantial demonstrations of CCA-based applications for use in outreach activities.

Application Focus: Chemistry

Investigators: D. Bernholdt (ORNL), C. Janssen (SNL), T. Windus (PNNL)
Collaborators: R. Kendall (Ames Lab), A. Wagner (ANL)

Computational chemistry is fundamental to DOE's science mission, from energy efficiency to combustion modeling to waste cleanup and environmental remediation. Terascale computing presents new opportunities in chemistry, not only to address larger problems, but also to begin linking models representing different aspects of computational chemistry (i.e., molecular dynamics, electronic structure, and kinetics) or different length scales (atomistic calculations, chemical mechanisms, and direct numerical simulation). Because particular codes are often very large and complex (for example, NWChem [NWChem01] is in excess of 90,000 lines long and represents more than 100 person-years of development effort [Dunning00]), linking them in order to do new science represents a tremendous challenge. At present, doing so with any measure of generality is quite unrealistic. Component technology, however, can make such a problem tractable.

Our work with chemistry application software will focus on developing interface specifications and components that will offer novel and domain-relevant demonstrations of the power and generality of CCA in creating linkages as described above. At the same time, we will provide the Center with important feedback in practical applications. This work also provides a model for other chemistry groups using CCA to make similar linkages [Harrison01,Hewson01,Najm01,Trouve01]. It will provide a vehicle to investigate issues regarding multiple implementations of a given component: how they can be described, chosen, and used in situations where implementations and behavior may differ significantly at a detailed level (similar to the "quality of service" ideas discussed in the *Parallel Components* Section).

We will begin by developing high-level interfaces among several electronic structure methods in NWChem [NWChem01] and MPQC [Janssen95] (e.g., SCF, DFT, MP2, and coupled cluster) and ANL's Toolkit for Advanced Optimization (TAO). This work will provide componentized tools to optimize molecular structures at given levels of theory, a capability central to computational chemistry. NWChem and MPQC represent the state of the field for massively parallel computers and were developed using object-oriented techniques, which will facilitate componentization. While both packages already include several optimization methods, the use of the TAO component will provide "instant" access to the latest optimization techniques, thereby freeing the chemists to focus on chemistry. The development of general high-level interfaces also enables a degree of interoperability between the two electronic structure packages, so that users will no longer be limited to the set of methods offered by one specific package.

These high-level interfaces have general utility in this domain and can be used in many more complex applications. We will explore several such applications in greater depth in the second phase of this work. One area is the determination of protein/ligand binding sites and affinities, which has uses ranging from drug development to studying the effects of toxins on cells. Protein/ligand binding studies involve the use of progressively more accurate and more expensive methods to sieve potential ligands from a library until a sufficiently small set remains for experimental validation. Using current software approaches, users are often forced to accept the limitations of their chosen packages because of the lack of interoperability. By using CCA technology, it becomes straightforward to provide access to a wide range of methods from numerous packages. The SNL team will create a CCA-based tool for automated protein/ligand binding studies, utilizing the components developed during the first phase (from MPQC, NWChem, and TAO), plus additional methods such as molecular mechanics/dynamics, linked by a new driver code.

A second area of application is the use of electronic structure packages for the direct computation of chemical kinetics. For example, a proposal to the SciDAC Computational Chemistry Program lead by A. Wagner [Wagner01] brings new levels of generality, rigor, and automation to the problem. This work involves an unprecedented degree of coupling of electronic structure and kinetics codes, as well as molecular dynamics and new software for discovering and mapping critical regions of the molecular potential energy surface (PES). We will focus initial efforts on the DIRect DYnamics package, DIRDY [DIRDY], which gathers electronic structure data for POLYRATE [Corchado00], a variational transition state theory kinetics program. We will extend the interfaces developed in phase one to link the electronic structure packages and DIRDY. This work will build on both the code and the experience gained at PNNL in developing the existing "one-off" interface between DIRDY and NWChem. We will demonstrate the generality of this interface by using it to link DIRDY with MPQC, which does not currently have such an interface. Our collaborators at Ames Lab plan to implement the same interface for the GAMESS electronic structure code [Schmidt93], which has a one-off interface with DIRDY. Since GAMESS development began in the late 1970s, it provides a good test of the use of CCA in older applications. Finally, we will use these components and others developed in this proposal (see the *Parallel Components* Section) as the basis for their new PES discovery/mapping capability, which will be built CCA-compliant from scratch.

The work described so far targets interoperability at fairly high levels. In the third phase of this effort, we will examine the use of CCA at a finer granularity. This research is important in understanding how to best use component technologies in scientific software. It also offers the potential to completely revolutionize the development of software in fields like chemistry. At present, there are numerous large computational chemistry packages with significant overlaps in functionality. But packages typically specialize, offering more computational features than other codes in certain areas, and users are essentially locked in to the capabilities of a given package. Creating standardized interfaces among low-level chemistry components opens the way to creating truly interoperable software and ultimately to creating custom tailored applications.

We will examine this kind of interoperability first in the evaluation of molecular properties. We will develop interfaces that allow MPQC, which currently has only very basic property capabilities, to utilize NWChem's fairly extensive property capabilities. We will also investigate the more complex case of solvation models. Solvation models attempt to incorporate important effects of the solvent environment common in real-world chemical problems into an electronic structure calculation at a much lower cost than treating the solvent explicitly. GAMESS provides a number of sophisticated solvent models, while NWChem offers one widely used model and MPQC has none. Interoperability is more complex in this case because the solvent model is more intricately interconnected to other aspects of the calculation. This example will require more extensive involvement of our collaborators at Ames Lab, which may be facilitated by direct support from the Center if additional funding becomes available in later years. If this is not possible, we will chose another low-level interoperability example that can be carried out with the original team.

Our proposed work with chemistry applications will provide novel CCA-based computational capabilities to that community, while at the same time providing important feedback to the development of the CCA itself. We will pioneer the development of general component-based interfaces in this domain and gain valuable experience with the integration of CCA into large, complex, pre-existing software packages and the use of third-party numerical components. Through this work, culminating in the low-level interoperability studies, we hope to catalyze a major change in the way computational chemistry software is produced throughout the community.

Application Focus: Climate Modeling

Investigators: D. Bernholdt (ORNL), J. Larson (ANL)
Collaborators: J. Drake (ORNL), C. DeLuca (NCAR)

Computational climate modeling is critical to world-wide efforts in understanding the Earth's climate and effects of anthropogenic activities. DOE makes a substantial investment in scientific research in climate modeling and simulation through its Climate Change Prediction Program, and DOE researchers are integral to the national community, which develops, tests, and uses computational climate models. Increasing computational power has allowed the climate modeling community to improve the sophistication, fidelity, and resolution of their models. More recently, it has become practical to begin coupling models representing different components of the climate system (i.e., atmosphere, ocean, ice, land, biogeochemistry, etc.) to exchange time-dependent boundary data, thereby allowing more complex and realistic models of the Earth's climate system to be created. This activity is at a relatively early stage in both scientific and computational terms. Computationally, this community is trying to address in a specific context many of the same questions facing the CCA group in a more general context, providing an excellent opportunity for a two-way dialog.

In addition, NASA will soon begin to create an Earth System Modeling Framework (ESMF), an open framework based on standardization of interfaces into which climate modeling and related software can be "plugged". The Common Component Architecture has been discussed in several proposals to NASA's ESS/HPCC program as a logical basis on which to build the ESMF. The chance to become involved at the beginning of the ESMF activity represents a significant opportunity for the Common Component Architecture. Throughout the course of this work, we will pay close attention to the evolution of activities like the Earth System Modeling Framework and adjust the scope and focus of this effort in order to have maximum impact on and input into their development.

The Center's work in the climate modeling domain will be closely related with the work done with the Community Climate System Model (CCSM) [Boville98] under the current Accelerated Climate Prediction Initiative Avant Garde (ACPI-AG) effort and the proposed follow-on [Malone01]. Investigator Larson, and our ORNL and ANL collaborators in this work, are all members of the CCSM's Software Engineering Working Group (SEWG). One of the results of the ACPI-AG project has been the development of the Model Coupling Toolkit (MCT) [MCTa,MCTb]. The MCT provides a set of low-level application program interfaces (APIs) to describe data fields, the grids on which they are evaluated, and their parallel decompositions. The MCT also provides interfaces that support some of the basic operations required in coupling: transformation of grids, accumulation and merging of data fields, etc. On top of this, a "coupler" is being developed, which involves a combination of high-level management/control of the multiple simulation components and the lower-level data management/movement functionality [Kauffman97,Valcke00].

We will work closely with the parallel data redistribution and scientific data component efforts (see the *Parallel Components* and *Parallel Data Redistribution* sections) to develop general CCA interface specifications incorporating experience from the MCT in these areas. Then, in step with CCA implementations of these interfaces, we will begin incrementally replacing existing MCT functionality with CCA-based functionality. We will retain the MCT interface as far as possible, providing adapters to the more general interface of the CCA components underneath. The ability of CCA components to adapt to a rather complex existing interface such as MCT will serve as an important test of their usability. Then, working together with the CCSM community, we will begin exposing the direct CCA interfaces to the various climate components and taking advantage of other CCA components, like the data interface broker (see the *Parallel Components* section). This work will provide an opportunity to examine the relative merits of the general CCA-based interface and the domain-specific MCT interface in terms of their relative efficiency for software development and performance. This experience will provide valuable guidance on how other projects might approach adoption the CCA MxN redistribution/coupling capability.

Model coupling is just one of the many opportunities for CCA in climate modeling systems. Other areas of interest include model diagnostic tools, I/O, and data assimilation. Component technologies can also be used within individual models, for example linking the physics and dynamics aspects of atmospheric models. In the later years of this work, we will expand our activities to include one or more of these areas based on an assessment of where we can have the most impact at that time. As part of this broader view of the field, we will also work with ETCs focusing on climate [Drake01] and meshing [Glimm01] issues to insure that their software is compatible with CCA and, where appropriate, to incorporate it into this work at the earliest opportunity.

Other Applications Areas

While we have cited specific proposals to help motivate the focused efforts above, the work described is more general in nature, and it will provide examples and practical experience that can be applied to other projects adopting CCA. As mentioned, we expect to consult with the majority of projects using CCA technology, but these projects also will be very important to the development of the CCA, and therefore will be closely followed. Based on proposals and projects of which we are currently aware (see Appendix C), we expect these applications to include:

- Use of CCA to develop frameworks for research in reacting flows and turbulent combustion
- Use of CCA as the framework for problem solving environments in multiscale chemical science, macromolecular x-ray crystallography, bioengineering, and semiconductor device simulation
- Use of CCA in the creation of an integrated accelerator simulation environment
- Development of components and interfaces to facilitate integration of CCA-based applications into collaborative environments and problem solving environments
- Development of CCA-compliant software components in many ETCs

Deliverables Summary

We present a brief statement highlighting proposed Center research at each institution, followed by a work breakdown summary. Work during the proposal's out-years will likely shift somewhat among topics in the work breakdown summary, as various projects come to fruition and new research issues arise. Finally, we present deliverables timelines for research in frameworks, components, parallel data redistribution, and applications integration.

The following summary highlights individual laboratory and university contributions to the proposed Center:

- **ANL** will collaboratively define abstract interface suites for various parallel numerical components and will develop CCA-compliant implementations for mesh management, linear solvers, nonlinear solvers, optimization software, and low-level services; ANL will also investigate issues in numerical component quality-of-service and will introduce CCA technology into climate model coupling.
- **Indiana University** (IU) will integrate the CCA standard framework with the appropriate distributed computing protocols and services so that CCA components will be able to run as distributed applications on the emerging DOE grid; IU will also develop CCA-compliant linear solver components.
- **LANL** will collaboratively explore issues in collective interactions between collective components, in particular, to develop generic data redistribution components to enable the exchange of distributed data between components.
- **LLNL** will contribute in the areas of language interoperability technology for scientific programming languages (Fortran 77, Fortran 90, C, C++, Java, Python, MATLAB, and others), the Alexandria component repository to deploy component software, and parallel data redistribution research.
- **ORNL** will collaboratively design a specification for parallel data redistribution and develop a corresponding CCA-compliant implementation as well as components for interactive visualization, computational steering, and fault-tolerance; ORNL will also contribute to parallel data component research and act as a liaison for application integration in various domains, including climate and chemistry.
- **PNNL** will contribute to parallel data component research and will investigate the use of CCA technology in chemistry applications.
- **SNL** will develop and deploy a parallel CCA-compliant framework called CCAFFEINE, complete with a component deployment environment and components for adaptive structured meshes, multi-threading, and load redistribution.
- **The University of Utah** (UU) will develop a standard CCA graphical user interface environment, define standards for multi-threaded frameworks, and investigate framework-based methods for parallel component interactions.

Work Breakdown Summary (with level of effort given in FTE units)

Institution	Frameworks		Parallel Components		MxN		Applications		Total
ANL	Low-level services	0.5	Data Components Optimization Nonlinear Solvers	0.8 0.7 0.5			Climate	0.5	3.0
IU	Distributed Framework	1.0	Linear Solvers	0.2					1.2
LANL					Component	0.5			0.5
LLNL	Language Interoperability Component Repository	1.0 0.5			Framework	1.0			2.5
ORNL			Fault Tolerance Visualization & Steering	0.5 0.5	Component	1.0	Climate Liaison	0.75 0.25	3.0
PNNL			Data Components	0.5			Chemistry	0.5	1.0
SNL	SCMD Framework	1.0	Data Components	0.5			Chemistry	1.0	2.5
UU	Builder Service and Thread Safety	0.3	GUI Component	0.2	Framework	0.5			0.8
Total FTEs		4.3		4.4		2.8		3.0	14.5

Frameworks Deliverables

'01	'02	'03	'04	'05
Complete candidate SCMD build standard and implementation (SNL)	Adapt strict SCMD concepts to more general scheme for HPC (SNL)	Improve dynamic user interaction of SCMD framework (SNL)	Iterate on design and improve per user requirements (SNL)	Iterate on design and 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 applications 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 (LLNL)	Deploy Alexandria component repository and assist collaborators with integrating components (LLNL)		
Complete Babel language interoperability support for Python and Java (LLNL)	Add Babel language interoperability support for Fortran 90 (LLNL)	Add Babel language interoperability support for MATLAB (LLNL)	Add Babel support for Perl or other languages as needed (LLNL)	Add Babel support for COM and dot-NET for Windows (LLNL)
Complete basic concurrency standard and implementation (all)	Evolve concurrency design per user requests (all)			
Develop a distributed multiplexer and XML protocol prototype (IU,LLNL,SNL)	Iterate on SCMD/Grid multiplexer per user requirements (IU, SNL)			

Parallel Components Deliverables

'01	'02	'03	'04	'05
Evaluate application requirements (all)	Deploy prototypes, evaluate & refine with apps feedback (all)	Evaluate, refine, and extend components; offer tutorials (all)	Evaluate, refine, and extend components; offer tutorials (all)	Evaluate, refine, and extend components; offer tutorials (all)
Develop prototype components for 2D/3D visualization (ORNL)	Develop prototype computational steering component (ORNL)	Integrate viz components with data components (ORNL)	Extend viz support to desktop (ORNL, UU)	Extend viz to un-structured, adaptive data (ORNL, ANL)
Develop prototypes for multithreaded (SNL) & GUI (UU) components	Develop load redistribution comp; extend multi-threaded & GUI comp (SNL,UU)	Use GUI comp for viz & numerical comps (ANL, ORNL, UU)		
	Add port fault notification and hooks into SCMD event service (ORNL, SNL)	Develop prelim SCMD fault monitoring & recovery component (ORNL, SNL)	Incorporate fault notification hooks into Grid event service (IU, ORNL)	Add distrib support to fault monitoring & recovery comp (IU, ORNL)
Develop prototype components for linear & nonlinear solvers, optimiz., & low-level services (ANL, IU, with TOPS center)	Define interfaces between MPQC, NWChem, & TAO (ANL, PNNL, SNL); develop prototype component factory for Jacobians/Hessians (ANL)	Investigate app-specific comps in optimiz. (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 center)	Circulate new data specs; deploy components supporting adaptivity (ANL, ORNL, PNNL, SNL, with TSTT center)	Add support for hybrid schemes (ANL, ORNL, PNNL,SNL, with TSTT center)
Develop prototype data interface broker & transformation comp (SNL)	Extend interface broker to work automatically; deploy SAMR component in combustion app (SNL)	Perform initial high-fidelity 3D combustion runs using SAMR component (SNL)	Test SAMR component with complex flame simulation (SNL)	

Parallel Data Redistribution Deliverables

'01	'02	'03	'04	'05
Develop prototype MxN parallel data exchange component for structured, rectilinear meshes, based on CUMULVS technology (ORNL, SNL)	Develop preliminary data translation and interpolation components for model coupling (ORNL, SNL)	Develop generalized MxN parallel data exchange component for structured, rectilinear meshes (LANL, ORNL)	Evaluate and tune MxN parallel data exchange component in climate and chemistry applications (LANL, ORNL)	Develop generalized MxN parallel data exchange component for structured, unstructured, & adaptive meshes, based on the scientific data comp (all)
Develop prototype MxN parallel data exchange component for structured, rectilinear meshes, based on PAWS technology (LANL)	Integrate ARMCI into MxN parallel data exchange component (ORNL, PNNL)			Experiment with composite data redistribution components, optimized for performance (LANL)
Investigate preliminary programming models for parallel remote method invocation (PRMI) (LANL, LLNL)	Add PRMI to Babel language interoperability tool (LLNL)	Extend PRMI system to support parallel data arguments via the scientific data component (LANL, LLNL)	Investigate SOAP performance issues protocol & improve performance for data redistribution (LLNL)	

Application Integration Deliverables

'01	'02	'03	'04	'05
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)	Integrate data & MxN components into MCT (ANL, ORNL)	Integrate MxN data transformation components into MCT (ANL, ORNL)	Evaluate and tune CCA-MCT and MxN components; expand CCA beyond base MCT (e.g., model diagnostics, data assimilation) (ANL,ORNL)	Complete prototype of full climate coupler using CCA (ANL, ORNL)

Software Evolution and Distribution

This project will involve the creation of frameworks, components, and related software by all of the Center members. Further, we expect many outside groups use our software and contribute their own modifications. At the earliest reasonable point, we will establish a CVS software repository available to Center members and collaborators. Such a CVS repository has already been established that contains the CCA specification, the CCAFFEINE framework, and CUMULVS-based visualization components. Eventually, as the software matures, we expect to open access to a larger community through anonymous CVS (checkout only). This type of access will serve primarily developers of core software.

When tools and components reach a level of maturity consistent with general deployment, we will make them available through the web-based repository, Alexandria [Alexandria01], which is under development at LLNL as part of the Center. Alexandria will enable CCA software and documentation to be made available through a single unified site to both human users and to automated tools, such as graphical component builders. Component developers will ultimately be responsible for user support for their software. The Center will setup a web-based "trouble ticket" or “bug tracking” system to help ensure that support requests are tracked and answered. The CCA web site [CCA01] will be the primary point of dissemination for CCA software and documentation, and it will link to the appropriate CVS and Alexandria repositories.

Finally, we will need to address licensing issues for collaborative software development by proposal participants, as each laboratory and university has its own intellectual property rules and restrictions. It is our desire to release our software using one of the approved open source licenses [OS]. We hope that the Office of Science will provide guidance to DOE laboratories with respect to software release issues for collaboratively developed software intended for the general research community.

Subcontract or Consortium Arrangements

Appendix C summarizes our proposed collaborations with other SciDAC ETCs, SciDAC application groups, and other efforts. Appendix C also includes letters of support from ETC and application groups.

References

[Abate00] J. Abate, S. Benson, L. Grignon, P. Hovland, L.C. McInnes, and B. Norris, "Integrating Automatic Differentiation with Object-Oriented Toolkits for High-Performance Scientific Computing," Argonne preprint ANL/MCS-P820-0500, 2000, to appear in the *Proceedings of the AD2000 Conference*, Springer.

[Alexandria01] Alexandria Web Page, https://www-casc.llnl.gov/alexandria.

[ALICE01] ALICE Web Page, http://www.mcs.anl.gov/alice.

[Allan01] B.A. Allan, R.C. Armstrong, A.P. Wolfe, J. Ray, D.E. Bernholdt, and J.A. Kohl, "The CCA Core Specification In a Distributed Memory SPMD Framework," submitted to *Concurrency: Practice and Experience*, 2001.

[Anderson99] W.K. Anderson, W.D. Gropp, D.K. Kaushik, D.E. Keyes, B.F. Smith, "Achieving High Sustained Performance in an Unstructured Mesh CFD Application," *Proceedings of SC99*, also available as Argonne preprint ANL/MCS-P776-0899.

[Atlas95] S. Atlas, S. Banerjee, J.C. Cummings, P.J. Hinker, M. Srikant, J.V.W. Reynders, and M. Tholburn, "POOMA: A High Performance Distributed Simulation Environment for Scientific Applications," *Proceedings of Supercomputing '95*, December 1995. See http://www.acl.lanl.gov/POOMA.

[Armstrong99] 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," *High Performance Distributed Computing Conference*, 1999. See http://www.cca-forum.org.

[Balay97] S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith, "Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries," in *Modern Software Tools in Scientific Computing*, 1997, E. Arge, A.M. Bruaset and H.P. Langtangen, Ed., Birkhauser Press, pp. 163-202.

[Balay98] S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith, "A Microkernel Design for Component-based Parallel Numerical Software Systems," *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*, 1998, pp. 60-69, also available as Argonne preprint ANL/MCS-P727-0998.

[Balay00] S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith, *PETSc 2.0 Users Manual*, MCS Technical Report, ANL-95-11, Revision 2.0.29, November 2000. See http://www.mcs.anl.gov/petsc.

[Beck99] M. Beck, J. Dongarra, G. Fagg, A. Geist, P. Gray, J. Kohl, M. Migliardi, K. Moore, T. Moore, P. Papadopoulos, S. Scott, V. Sunderam, "HARNESS: A Next Generation Distributed Virtual Machine," *Future Generation Computer Systems*, Special Issue on Metacomputing, Elsevier, Volume 15, Numbers 5/6, 1999.

[Beckman98] P. Beckman and P. Fasel and W. Humphrey and S. Mniszewski, "Efficient Coupling of Parallel Applications using PAWS," *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computation*, July 1998. See http://www.acl.lanl.gov/paws.

[Benson00] S. Benson, L.C. McInnes, and J.J. Moré, "GPCG: A Case Study in Parallel Optimization Software," Argonne preprint ANL/MCS-P768-0799, 2000, submitted to *ACM Transactions on Mathematical Software*.

[Benson00b] S. Benson, L.C. McInnes, and J.J. Moré, *TAO Users Manual*, MCS Technical Memorandum, ANL/MCS-TM-242, Revision 1.0.2, November 2000. See http://www.mcs.anl.gov/tao.

[Bischof96] C. Bischof, A. Carle, P. Khademi, and A. Mauer, "ADIFOR 2.0: Automatic Differentiation of Fortran 77 Programs," *IEEE Computational Science and Engineering*, number 3, volume 3, 1996, pp. 18-32.

[Bischof97] C. Bischof, L. Roh, and A. Mauer, "ADIC: An Extensible Automatic Differentiation Tool for ANSI-C," *Software Practice and Experience*, number 12, volume 27, 1997, pp. 1427-1456.

[Box00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," *W3C Note 08*, May 2000. See http://www.w3.org/TR/-2000/NOTE-SOAP-20000508.

[Boville98] B. A. Boville and P. R. Gent, "The NCAR Climate System Model, Version One," *J. Climate*, 11, 1115-1130, 1998.

[Bramley98] R. Bramley, D. Gannon, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, and M. Govindaraju, "Component Architectures for Distributed Scientific Problem Solving," *IEEE Computational Science and Engineering*, 5, no. 2 (1998) pp.50-63.

[Bramley99] R. Bramley, D. Gannon, J. Villacis, A Whitaker, "Using the Grid to Support Software Component Systems," *Proceedings of the 1999 SIAM Parallel Processing Conference*.

[Bramley00] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, M. Yechuri, "A Component Based Services Architecture for Building Distributed Applications," *Proceedings of HPDC*, 2000.

[Brown99] D.L. Brown, W.D. Henshaw, and D.J. Quinlan, "Overture: An Object-Oriented Framework for Solving Partial Differential Equations on Overlapping Grids," *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*, SIAM, 1999, pp. 215-224.

[Browne00] J.C. Browne, E. Berger, and A. Dube, "Compositional Development of Performance Models in POEMS," *The International Journal of High Performance Computing Applications*, vol. 14, number 4, pp. 283-291, 2000.

[Buschelman00] K.R. Buschelman, W.D. Gropp, L.C. McInnes, and B.F. Smith, "PETSc and Overture: Lessons Learned Developing an Interface between Components," Argonne preprint ANL/MCS-P858-1100, 2000, to appear in the *Proceedings of the International Federation for Information Processing Working Conference on Software Architectures for Scientific Computing*, Kluwer.

[CCA01] The Common Component Architecture Technical Specification, Version 0.5. See http://www.cca-forum.org.

[Chow98] E. Chow, A. Cleary, R. Falgout, "Design of the *hypre* Preconditioner Library," *Proceedings of the SIAM Workshop on Object-Oriented Methods for Inter-Operable Scientific and Engineering Computing*, Yorktown Heights, NY, October 21-23. See http://www.llnl.gov/CASC/hypre.

[Cleary98] A. Cleary, S. Kohn, S. Smith, B. Smolinski, "Language Interoperability Mechanisms for High-Performance Scientific Applications," *Proceedings of the SIAM Workshop on Object-Oriented Methods for Inter-Operable Scientific and Engineering Computing*, Yorktown Heights, NY, October 21-23, 1998.

[Davison00] J. Davison de St. Germain, J. McCorquodale, S.G. Parker, and C.R. Johnson, "Uintah: A Massively Parallel Problem Solving Environment," *IEEE Int. Symp. High Perform. Distrib. Comput. Proc.*, IEEE, Piscataway, NJ, pp. 33-41, 2000.

[de Sturler00] E. de Sturler, J. Hoeflinger, L. Kale, M. Bhandarkar, "A New Approach to Software Integration Frameworks for Multi-physics Simulation Codes," *Proceedings of the International Federation for Information Processing Working Conference on Software Architectures for Scientific Computing*, Kluwer, 2000, to appear.

[DIRDY] DIRect DYnamics, Bruce C. Garrett, Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory; Yao-Yuan Chuang and Donald G. Truhlar, Department of Chemistry and Super Computer Institute, University of Minnesota.

[Drake01] J. Drake et al., "Mathematical and Algorithmic Aspects of Climate Dynamics", to be submitted to the SciDAC Integrated Software Infrastructure Centers program (01-07).

[Dunning00] Thom H. Dunning, Jr., private communication, 2000.

[Eddon98] G. Eddon and H. Eddon, *Inside Distributed COM*, Microsoft Press, 1998.

[Epperly00] T. Epperly, S. Kohn, and G. Kumfert, "Component Technology for High-Performance Scientific Simulation Software," *Proceedings of the International Federation for Information Processing Working Conference on Software Architectures for Scientific Computing*, Kluwer, 2000, to appear.

[ESI] Equation Solver Interface (ESI) Web Page, http://z.ca.sandia.gov/esi.

[Fingar00] P. Fingar, "Component-Based Frameworks for E-Commerce," *Comm. ACM* 43(10) October 2000.

[Freitag98] L. Freitag, M. Jones, and P. Plassmann, "The Scalability of Mesh Improvement Algorithms," *Algorithms for Parallel Processing*, ed. M. Heath, A. Ranade, and R. Schreiber, *IMA Volumes in Mathematics and Its Applications*, Volume 105, Springer-Verlag, 1998, pages 185-212.

[Freitag98b] L. Freitag, M. Jones, and P. Plassmann, "Mesh Components and Software Integration with SUMAA3d," *Proceedings of the SIAM Workshop on Object-Oriented Methods for Inter-Operable Scientific and Engineering Computing*, Yorktown Heights, NY, October 21-23, 1998, pages 215-224.

[Freitag99] L. Freitag, *Users Manual for Opt-MS: Local Methods for Simplicial Mesh Smoothing and Untangling*, ANL Technical Report, Number ANL/MCS-TM-239, Argonne National Laboratory, Argonne, Illinois, March, 1999.

[Freitag99b] L.A. Freitag, W.D. Gropp, P.D. Hovland, L.C. McInnes, and B.F. Smith, "Infrastructure and Interfaces for Large-Scale Numerical Software," *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999, H. R. Arabnia, ed., pp. 2657-2664, also available as Argonne preprint ANL/MCS-P751-0599.

[Freitag00] L. Freitag and P. Plassmann, "Local Optimization-based Simplicial Mesh Untangling and Improvement," *International Journal of Numerical Methods in Engineering*, vol. 49, issue 1-2, July, 2000, pp. 109-125.

[Finn01] J. Finn et al., "Center for Scalable, Implicit, Nonlinear, Extended Magnetohydrodynamics," proposal to be submitted to the SciDAC Fusion Energy Program (01-10).

[Gamma95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Gannon98] D. Gannon, and A. Grimshaw, "Object-Based Approaches," *The Grid: Blueprint for a New Computing Infrastructure*, Ian Foster and Carl Kesselman (Eds.), pp. 205-236, Morgan-Kaufman, 1998.

[Geist97] G. A. Geist, J. A. Kohl, P. M. Papadopoulos, "CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications," *Int. Journal of High Performance Computing Applications*, Volume 11, Number 3, August 1997, pp. 224-236.

[Geist99] G. A. Geist, J.A. Kohl, P.M. Papadopoulos, S. L. Scott, "Beyond PVM 3.4: What We've Learned, What's Next, and Why," *Future Generation Computer Systems*, special issue on Metacomputing, Elsevier Publishing, Volume 15, Numbers 5/6, 1999, pp. 571-582.

[GF] See the Grid Forum web pages at http://www.gridforum.org.

[Glimm01] J. Glimm et al., "Terascale Simulation Tools and Technologies Center," to be submitted to the SciDAC Integrated Software Infrastructure Centers Program (01-07).

[Globus] See the Globus home page at http://www.globus.org.

[Govindaraju00] M. Govindaraju, A. Slominski, V. Chopella, R. Bramley, and D. Gannon, "Requirements for Evaluation of RMI Protocols for Scientific Computing," *Proceedings of SC00*, November 2000.

[Griewank00] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Philadelphia, 2000.

[Gropp00] W.D. Gropp, D.E. Keyes, L.C. McInnes, and M.D. Tidriri, "Globalized Newton-Krylov-Schwarz Algorithms and Software for Parallel Implicit CFD," *Int. Journal on High Performance Computing Applications*, volume 14, pp. 102-136, 2000.

[Guyer99] S. Guyer, "An Annotation Language for Optimizing Software Libraries," *2nd Conference on Domain Specific Languages*, Oct. 1999, pp. 39-53.

[Harrison01] R.J. Harrison et al., "Advanced Methods for Electronic Structure," proposal to be submitted to the SciDAC Computational Chemistry program (01-08).

[Hewson01] J.C. Hewson et al., "Collaboratory for Multi-scale Chemical Science," proposal to be submitted to the SciDAC National Collaboratories program (01-06).

[Hindmarsh01] A. Hindmarsh et al., PVODE Web Page, http://www.llnl.gov/CASC/PVODE.

[Hollingsworth98] J. K. Hollingsworth and P. J. Keleher, "Prediction and Adaptation in Active Harmony," *The Seventh International Symposium on High-Performance Distributed Computing*, July 1998, pp. 180-188.

[Hornung98] R. Hornung and S. Kohn, "The Use of Object-Oriented Design Patterns in the SAMRAI Structured AMR Framework," *Proceedings of the SIAM Workshop on Object-Oriented Methods for Inter-Operable Scientific and Engineering Computing*, October 1998. See http://www.llnl.gov/CASC/SAMRAI.

[Hornung01] R. Hornung and S. Kohn, "Managing Application Complexity in the SAMRAI Object-Oriented Framework," *Concurrency and Computation: Practice and Experience* (special issue on Software Architecture for Scientific Applications), to appear, 2001.

[Illinca97] F. Illinca, J.-F. Hetu, and R. Bramley, "Simulation of 3D Mold-Filling and Solidification Processes on Distributed Memory Parallel Architectures," *Proceedings of the International Mechanical Engineering Congress and Exposition*, 1997.

[Isis01] ISIS++ Web site. See http://z.ca.sandia.gov/isis.

[Janssen95] C.L. Janssen, E.T. Seidl, and M.E. Colvin, "Object-Oriented Implementation of Parallel Ab Initio Programs," in ACS Symposium Series 592, *Parallel Computers in Computational Chemistry*, T. Mattson, Ed., 1995.

[Janssen96] C.L. Janssen, E.T. Seidl, I.M.B. Nielsen, and M.E. Colvin, *The Scientific Computing Matrix Library*, Sandia Report SAND97-8200 UC-405, Sandia National Laboratories, 1996.

[Jardin01] S. Jardin et al., "Center for Extended Magnetohydrodynamic Modeling," proposal to be submitted to the SciDAC Fusion Energy Program (01-10).

[Johnson95] C.R. Johnson and S.G. Parker, "Applications in Computational Medicine using SCIRun: A Computational Steering Programming Environment," *Supercomputer `95*, H.W. Meuer ed., pp. 2-19, Springer-Verlag, 1995.

[Kauffman97] B. Kauffman, "The NCAR CSM Flux Coupler, Version 4.0," NCAR Technical Note, NCAR/TN-424+STR, June 1997.

[Keahey97] K. Keahey and D. Gannon, "PARDIS: A Parallel Approach to CORBA," *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computation*, August 1997.

[Keahey97b] K. Keahey and D. Gannon, "PARDIS: CORBA-based Architecture for Application-Level PARallel DIStributed Computation," *Proceedings of Supercomputing '97*, November, 1997.

[Keahey98] K. Keahey and D. Gannon, "Developing and Evaluating Abstractions for Distributed Supercomputing," *Journal of Cluster Computing*, special issue on High Performance Distributed Computing, Vol. 1, No. 1, May 1998.

[Keahey00] K. Keahey, P. Beckman, and J. Ahrens, "Ligature: Component Architecture for High-Performance Applications," to appear in *The International Journal of High Performance Computing Applications*, Winter 2000, Vol. 14, No. 4 (Special Issue on Performance Modeling).

[Keller98] Ralph Keller and Urs Holzle, "Binary Component Adaptation," *Proceedings of ECOOP'98*, Brussels, July 1998.

[Kara99] D. Kara, "The Enterprise JavaBeans Component Model," *Component Strategies* 1(7), January 1999.

[Kohl97] J. A. Kohl, "High-Performance Computers: Innovative Assistants to Science," *ORNL Review*, Volume 30, Number 3 & 4, 1997, pp. 54-59.

[Kohl98] J. A. Kohl, P. M. Papadopoulos, "Efficient and Flexible Fault Tolerance and Migration of Scientific Simulations Using CUMULVS," Proceedings of the SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT 98), Welches, Oregon, August 3-4, 1998, pp. 60-71.

[Kohl99] J. A. Kohl, G. A. Geist, "Monitoring and Steering of Large-Scale Distributed Simulations," IASTED International Conference on Applied Modeling and Simulation, September 1-3, 1999, Cairns, Queensland, Australia.

[Kohn01] S. Kohn, G. Kumfert, J. Painter, and C. Ribbens. "Divorcing Language Dependencies from a Scientific Software Library," *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*, to appear 2001.

[LACSI] See the Los Alamos Computer Science Institute home page at http://lacsi.lanl.gov/research.

[Larsen00] G. Larsen, "Component-Based Enterprise Frameworks," Comm. ACM 43, 24--26 (2000).

[Malone01] B. Malone et al., “Comprehensive Design and Development of the Community Climate System Model for Tera-scale Computers,” to be submitted to the SciDAC Accelerated Climate Prediction Initiative (01-09).

[MCTa] Model Coupling Toolkit Web Page. See http://www.mcs.anl.gov/~larson/mct.

[MCTb] See http://www.mcs.anl.gov/~larson/mct/mct_APIs.pdf.

[Mousseau00] V.A. Mousseau, D.A. Knoll, and W.J. Rider, “Physics-based Preconditioning and the Newton-Krylov Method for Non-equilibrium Radiation Diffusion,” *J. Comput. Phys*, vol. 160, pp. 743-765, 2000.

[Najm01] H. Najm et al., "A Computational Facility for Reacting Flow Science", proposal to be submitted to the SciDAC Computational Chemistry program (01-08).

[NET] See the Microsoft dot-NET home page at http://www.microsoft.com/net.

[Nielsen00] I. M.B. Nielsen and C.L. Janssen, “Multi-threading: A New Dimension to Massively Parallel Scientific Computation,” *Comp. Phys. Comm*., 128:238, 2000.

[Nieplocha96] J. Nieplocha, R.J. Harrison, and R.J. Littlefield, “Global Arrays: A Nonuniform Memory Access Programming Model for High-Performance Computers,” *The Journal of Supercomputing*, 10:197-220, 1996. See http://www.emsl.pnl.gov:2080/docs/global.

[Nieplocha99] J. Nieplocha and B. Carpenter, “ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems,” *Proc. 3rd Workshop on Runtime Systems for Parallel Programming (RTSPP) of International Parallel Processing Symposium IPPS/SPDP '99*, San Juan, Puerto Rico, April 1999, in J. Rolim et al. (eds.) *Parallel and Distributed Processing*, Springer Verlag, LNCS 1586, 1999. See http://www.emsl.pnl.gov:2080/docs/parsoft/armci.

[NWChem01] See http://www.emsl.pnl.gov/pub/docs/nwchem.

[NWChem01b] R.J. Harrison, J.A. Nichols, T.P. Straatsma, M. Dupuis, E.J. Bylaska, G.I. Fann, T.L. Windus, E. Apra, J. Anchell, D. Bernholdt, P. Borowski, T. Clark, D. Clerc, H. Dachsel, W. deJong, M. Deegan, K. Dyall, D. Elwood, H. Fruchtl, E. Glendenning, M. Gutowski, A. Hess, J. Jaffe, B. Johnson, J. Ju, R. Kendall, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield, X. Long, B. Meng, J. Nieplocha, S. Niu, M. Rosing, G. Sandrone, M. Stave, H. Taylor, G. Thomas, J. van Lenthe, K. Wolinski, A. Wong, and Z. Zhang, "NWChem, A Computational Chemistry Package for Parallel Computers, Version 4.0.1" (2001), Pacific Northwest National Laboratory, Richland, Washington 99352-0999, USA.

[NWChem01c] R.A. Kendall, E. Apra, D.E. Bernholdt, E.J. Bylaska, M. Dupuis, G.I. Fann, R.J. Harrison, J. Ju, J.A. Nichols, J. Nieplocha, T.P. Straatsma, T.L. Windus, A.T. Wong, Computer Phys. Comm. 128, 2000, p. 260.

[OMG98] *The Common Object Request Broker: Architecture and Specification*, Object Management Group, February 1998. See http://www.omg.org/corba.

[OMG99] *CORBA Components*, Object Management Group, OMG TC Document orbos/99-02-95, March 1999. See http://www.omg.org.

[OMG01] See “CORBA Success Stories” at http://www.corba.org/success.htm.

[OS} See http://www.opensource.org/licenses for a list of approved open source licenses.

[Ousterhout98] J. Ousterhout, "Scripting: Higher Level Programming for the 21st Century," *IEEE Computer*, March 1998.

[Papadopoulos95] P. M. Papadopoulos, J. A. Kohl, "A Library for Visualization and Steering of Distributed Simulations using PVM and AVS," *High Performance Computing Symposium*, Montreal, Canada, July 10-12, 1995.

[Parashar00] M. Parshar and J. C. Browne, "System Engineering for High Performance Computing Software: The HDDA/DAGH Infrastructure for Implementation of Parallel Structured Adaptive Mesh Refinement," in IMA Volume 117: *Structured Adaptive Mesh Refinement*, pp. 1-18, Eds. S.B. Baden, M. P. Chrisochoides, D. B. Gannon and M. L. Norman, Springer-Verlag, Jan 2000. See also the Grid Adaptive Computational Engine Homepage at http://www.caip.rutgers.edu/~parashar/TASSL.

[Parker97] S.G. Parker, D.M. Weinstein, and C.R. Johnson, "The SCIRun Computational Steering Software System," *Modern Software Tools in Scientific Computing*, E. Arge, A.M. Bruaset, and H.P. Langtangen ed., Birkhauser Press, pp. 1-44, 1997.

[Parker97b] S.G. Parker, D.M. Beazley, and C.R. Johnson, "Computational Steering Software Systems and Strategies," *IEEE Computational Science and Engineering*, 1997.

[Parker99] S.G. Parker, *The SCIRun Problem Solving Environment and Computational Steering Software System*, Ph.D. Thesis, University of Utah, 1999.

[Pernice01] M. Pernice and M.D. Tocci, "A Multigrid Preconditioned Newton-Krylov Method," *SIAM J. Sci. Comput.*, 2001, in press.

[Corchado00] J. C. Corchado, Y.-Y. Chuang, P. L. Fast, J. Villà, W.-P. Hu, Y.-P. Liu, G. C. Lynch, K. A. Nguyen, C. F. Jackels, V. S. Melissas, B. J. Lynch, I. Rossi, E. L. Coitiño, A. Fernandez-Ramos, R. Steckler, B. C. Garrett, A. D. Isaacson, and D. G. Truhlar, POLYRATE version 8.5.1, University of Minnesota, Minneapolis, 2000.

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

[Quorum01] Quorum Web Page, http://www-casc.llnl.gov/quorum.

[René00] C. René, T. Priol, and G. Alléon, "Code Coupling Using Parallel CORBA Objects," *Proceedings of the International Federation for Information Processing Working Conference on Software Architectures for Scientific Computing*, Kluwer, to appear, 2000.

[Ribler99] R.L. Ribler, H. Simitci, and D.A. Reed, "The Autopilot Performance-Directed Adaptive Control System, Future Generation Computer Systems", special issue (Performance Data Mining), 1999.

[Schmidt93] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery, "General Atomic and Molecular Electronic Structure System," *J.Computat. Chem.*, volume 14, pp. 1347-1363, 1993.

[Smolinski99] B. Smolinski, S. Kohn, N. Elliott, and N. Dykman, "Language Interoperability for High-Performance Parallel Scientific Components," *International Symposium on Object-Oriented Parallel Environments (ISOPE)*, December 1999.

[Sparling00] Michael Sparling, "Lessons Learned Through Six Years of Component-Based Development," *Comm. ACM* 43, 47--53 (2000).

[Szyperski97] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, 1997.

[Trouve01] A. Trouve et al., "Terascale High-Fidelity Simulations of Turbulent Combustion with Detailed Chemistry," proposal to be submitted to the SciDAC Computational Chemistry program (01-08).

[Valcke00] S. Valcke, L. Terray, and A. Piacentini, *Oasis 2.4 User's Guide*, CERFACS Technical Note, CERFACS TR/CGMC/00-10, June 2000.

[Villacis99] J. Villacis, M. Govindaraju, D. Stern, A. Whitaker, F. Berg, P. Deuskar, B. Temko, D. Gannon, R. Bramley, "CAT: A High Performance, Distributed Component Architecture Toolkit for the Grid," *Proceedings of the High Performance Distributed Computing Conference*, 1999.

[W3C] *The Extensible Hypertext Markup Language*, World Wide Web Consortium. See http://www.w3c.org/TR/xhtml.

[Wagner01] A. F. Wagner et al., "Advanced Software for the Calculation of Thermochemistry, Kinetics, and Dynamics," proposal to be submitted to the SciDAC Computational Chemistry Program (01-08).

Appendix A: Proposal Management

In this section, we discuss issues associated with the management of this proposal. In particular, we address the research team, management structure, and budget.

Research Team

The proposal team includes participants from DOE laboratories (ANL, LANL, LLNL, ORNL, PNNL, and SNL) and two academic institutions (Indiana University and the University of Utah). Proposal participants have a long history of collaboration as members of the Common Component Architecture forum, which was formed in January of 1998 to develop component technology standards for the DOE. Since then, members of the CCA have met quarterly to discuss component technologies, jointly develop software, write and vote on component software specifications, write research papers, and advance the state-of-the-art in component technology for scientific computing.

Management Structure

Rob Armstrong will be the Principal Investigator for this proposal and will be the primary point of contact with DOE management. Four technical area leaders will support Armstrong: Scott Kohn (Component Framework), Lois Curfman McInnes (Component Software), James Kohl (Parallel Data Redistribution), and David Bernholdt (Application Integration). This management team will coordinate technical integration, monitor the project schedule, and report progress on deliverables during project reviews. These individuals, in addition to Dennis Gannon, Steve Parker, Jarek Nieplocha, and Craig Rasmussen, will serve as "line managers" of the project personnel at their respective institutions. We will also create an Advisory Committee consisting of Armstrong, the technical area leaders, and representatives from other ETC and applications groups to advise our Center on development priorities and user requirements. Members of the Center will meet quarterly as part of ongoing working meetings by the CCA forum, and subgroups will interact frequently via in-person visits and meetings using the Access Grid.

The proposal team has a history of collaborative research and development with Armstrong as leader of the Common Component Architecture forum. We have already developed a formal mechanism for proposing, voting, and adopting standards specifications using the LLNL Quorum voting server (http://www-casc.llnl.gov/quorum).

Appendix B: Facilities and Resources

Researchers in this proposal will require desktop computing resources and access to high-performance massively parallel computing system. Desktop resources are provided by individual institutions. High-performance computing resources are available at all facilities, and researchers will also have access to resources at other institutions. Some institutions provide Access Grid nodes for group-to-group collaboration environments, which will simplify communication among participants.

Argonne National Laboratory

Facilities in the Mathematics and Computer Science Division of Argonne National Laboratory include three major parallel computing systems, I/O subsystems, visualization subsystems, advanced display environments, collaborative environments and high capacity external network links. A 552-CPU Linux cluster, a 128-processor Silicon Graphics Origin 2000/Onyx 2 system with 12 Infinite Reality 2 graphics pipelines, and an 80-node IBM SP serve as the primary computation engines and are supported by hierarchical storage with approximately 2.5 TB of disk and a 60 TB tape robot. The SGI serves as the primary visualization server in addition to its use for large computational science experiments. All subsystems, as well as desktop workstations and various servers, are interconnected at gigabit ethernet speeds.

For high-end visualization, MCS maintains multiple immersive virtual reality devices including a 4-wall CAVE and 4 ImmersaDesks connected to a video-switching infrastructure that allows the display devices to be driven by either the SGI Onyx 2 or the Linux cluster. MCS also has three large-format mega-pixel tiled displays, one with ~11 million pixels, and two compact versions with ~3 million pixels each.

In addition, MCS currently supports four group-to-group collaboration environments (Access Grid nodes). The Access Grid is an ensemble of resources that supports group-to-group human interaction across the Internet. It consists of large-format multimedia displays, presentation and interactive software environments, interfaces to middleware, and interfaces to remote visualization environments. The Access Grid promotes group-to-group collaboration and communication for 3 to ~20 people per site with 2 to ~10 sites per session. Large-format displays integrated with intelligent or active meeting rooms are a central feature of Access Grid nodes.

Indiana University

The distributed systems research group has access to a wide range of workstations and several significant university research systems including a 64 processor Sun E10000, a 64 processor Linux cluster and a 184 processor IBM SP-2. We also have two HPSS auxiliary storage systems. This project will have access to all of these resources.

The departmental network consists of Gigabit Ethernet and 10Gbps aggregate bandwidth ATM backbones with workstation and server connectivity provided by a gigabit ethernet, 622Mbps ATM, 155Mbps ATM, 100Mbps fast ethernet, and 10Mbps ethernet connections. The department has a 100Mbps fast ethernet connection with the campus backbone, which provides high-speed access to university computing resources, including the Virtual Reality/Virtual Environment Facility (CAVE), and Internet, vBNS, and Internet2 network access. Network services available within the department include laser and color printing facilities, CD-ROM access and recording, scanners, video, and various media storage facilities.

Indiana University hosts the network operations center for the Abilene Network, which is the backbone for Internet 2. Consequently, we have excellent connections to all major, non-classified research networks.

Lawrence Livermore National Laboratory

Lawrence Livermore National Laboratory provides its computational researchers with desktop workstations, visualization servers, and high-performance computing resources. Researchers may access institutional resources such a Compaq AlphaServer cluster with 80 processors and 56GB of memory, an updated Compaq AlphaServer cluster with 136 processors and 80 GB of memory, a Linux cluster, a Sun Enterprise 6000 system, and associated high-performance storage resources. Researchers may also access the ASCI IBM massively parallel platform through arrangements with the ASCI program.

Los Alamos National Laboratory

Facilities in the Advanced Computing Laboratory (ACL) of Los Alamos National Laboratory provide access to a high-performance computing environment that includes:

- the largest unclassified computer in the world, the ACL Nirvana Machine, a 1-TeraOp peak speed SGI Origin 2000 system with one half terabyte of memory;
- the Little Blue Penguins cluster, a 128-node Pentium Pro cluster system running the Linux operating system and interconnected with Myrinet; and
- the largest unclassified, integrated visualization system in the world centered around 10 clustered SGI Infinite Reality systems attached directly to the ACL Nirvana Machine.

In addition, each investigator in the ACL is provided with a Linux desktop workstation attached to a 100-megabit ethernet network connecting to the rest of LANL and the outside world.

Oak Ridge National Laboratory

The Computer Science and Mathematics Division of ORNL operates substantial computer facilities, including desktop workstations, file and computer servers, all of which are connected to the ORNL site network. ORNL is connected to the DOE ESNet-3 through an OC-12 link. These will be the basic tools required to complete the research described.

The Computational Sciences (CCS) at the Oak Ridge National Laboratory provides state-of-the-art resources for high-performance computational science and computing science research. The primary computational resources currently include a 184-node, 724-processor IBM RS/6000 SP and a 64-node, 256-processor Compaq AlphaServer SC, each with over two terabytes (TB) of system-wide disk storage. Additional systems include a 16-node, 64-processor AlphaServer SC test system, a 32-processor SGI Origin for visualization, and various support servers. Center-wide storage is available in the form of the Distributed File System (DFS) and the High-Performance Storage System (HPSS), with 360 TB of archival storage. Internally, our systems are connected with gigabit ethernet. The Probe storage test bed, which provides additional high-end file servers and network bandwidth to other DOE laboratories augments this production infrastructure. The VizLab serves the CCS users=92 visualization needs with everything from charts and graphs to animations to fully immersive environments using our Immersadesk and Cave.

Planning is underway to upgrade the large AlphaServer SC to 128 nodes in the spring of 2001, which would bring the aggregate peak computational power of the CCS to 2 teraflops (TF). Long-range plans to field a 10-TF production system in 2003 are under discussion with DOE.

Pacific Northwest National Laboratory

Pacific Northwest National Laboratory provides its researchers with desktop workstations, visualization equipment, and high performance computing equipment. Access to institutional and external computing facilities is provided by high capacity network links.

Sandia National Laboratory

Sandia National Laboratory provides all necessary desktop workstations, visualization equipment, and high performance computing equipment, connected by an internal high capacity network. Access to external computing facilities is provided by external high capacity network links.

University of Utah

The Department of Computer Science Computing Facility is configured to support both instructional and research computing through a network of a few hundred workstations including Unix workstations from Hewlett-Packard, Silicon Graphics, IBM, Digital Equipment, and Sun, plus Windows NT (Intel) personal computers. These machines are supported by 30 file/application servers that provide 700 GByte of disk storage via both NFS file systems and 40 GByte of application replicated across five AFS servers. In addition to this full complement of workstations, the individual research laboratories contain a wide array of specialized equipment including a 32-processor SGI Origin 3000 with infinite reality graphics.

The Department of Computer Science also has a professional-quality video editing facility, a mobile robot, image capture and processing equipment, and a shared machine shop with a variety of NC milling machines/lathes/EDMs, and CMMs for CAD/CAM development.

Utah is the site of the first SGI Visual Supercomputing Center. The site is home to a 64 CPU Origin 2000, and a 40 CPU Onyx2 (with eight Infinite Reality graphics engines), and will be used in conjunction with the proposed desktop and cluster machines to carry out the proposed research.

The Scientific Computing and Imaging (SCI) Institute has ongoing research projects in many areas of scientific computing and imaging: geometric modeling, numerical analysis, parallel computing, problem-solving environments, scientific visualization, human-computer interaction, medical imaging and software development tools. The Institute is involved both in designing efficient and accurate tools for scientific computing, and in utilizing these tools for scientific applications.

Beyond the computing facilities within the SGI-Utah Visual Supercomputing Center, the SCI Institute has the following on-site facilities:

- SGI Origin 3000 (32 CPUs, 1 Infinite Reality graphics engine)
- SGI Power Onyx (14 CPUs, 2 RE2 graphics engines)
- SGI Origin 200 Server (4 CPUs)
- SGI Origin 200 Server (2 CPUs)
- Over 30 SGI Octanes, Indigo2s, O2s
- Numerous Linux PCs and Macs
- An Access Grid node
- 2.9 Terabyte RAID Array

Appendix C: Collaborators and Letters of Support

The following is a list of projects and proposals that plan to use Common Component Architecture technology and collaborate with the Center. We expect this to be a representative, but by no means complete, list of groups who will be working with us as they integrate CCA technology into their efforts. Principal investigators from these efforts have written letters expressing their interest in collaborating with the Center; these letters follow.

Title	Lead PI	Participating Institutions	Program
Terascale Optimal PDE Simulations (TOPS) Center	David Keyes (Old Dominion)	ANL, CMU, LBNL, LLNL, Tennessee, UC Boulder	SciDAC ISIC (01-07)
DOE Visualization Software Infrastructure Center	Rick Stevens (ANL)	ANL, LANL, LLNL, UC Davis, University of Utah	SciDAC ISIC (01-07)
Terascale Simulation Tools and Technologies Center	Jim Glimm (SUNY Stony Brook)	ANL, LLNL, ORNL, RPI, PNNL, SNL	SciDAC ISIC (01-07)
A High-Performance Software Framework and Interoperable Applications for the Rapid Advancement of Earth System Science: Part I: Core Earth System Modeling Framework Development	Tim Killeen (NCAR)	ANL, LANL, Michigan, MIT, NASA/GSFC, NCAR, NCEP, NOAA	NASA ESS/HPCC
A Grid-Based Collaboratory for Macromolecular X-Ray Crystallography Using Synchrotron Light Sources	Randall Bramley (Indiana)	ANL, Indiana	SciDAC Collaboratories (01-06)
A Computational Facility for Reacting Flow Science	Habib Najm, (SNL)	ANL, Colorado, FORTH (Greece), SNL, Terascale, UC Berkeley, UC Davis, U Roma (Italy)	SciDAC Chemistry (01-08)
Center for Scalable, Implicit, Nonlinear, Extended Magnetohydrodynamics	John M. Finn (LANL)	LANL, LLNL	SciDAC Fusion (01-10)
Increasing Interoperability of an Earth System Model: Atmosphere-Ocean Dynamics and Tracer Transports	Carlos R. Mechoso (UCLA)	LANL, NASA/JPL, LANL, UCLA	NASA HPCC/ESS
Shedding New Light on Exploding Stars: TeraScale Simulations of Neutrino-Driven Supernovae and their Nucleosynthesis	Anthony Mezzacappa (ORNL)	ORNL, SUNY, University of Chicago, University of Tennessee, Knoxville, NC State, University of Washington, Clemson, NCSA, UCSD, Florida Atlantic University, UIUC	SciDAC HENP (01-11)
Terascale High-Fidelity Simulations of Turbulent Combustion with Detailed Chemistry	Arnaud Trouve (Maryland)	Maryland, Michigan, Pittsburg Supercomputer Center, SNL, Wisconsin	SciDAC Chemistry (01-08)
Advanced Software for the Calculation of Thermochemistry, Kinetics, and Dynamics	Albert F Wagner (ANL)	ANL, Minnesota, Wayne State, SNL	SciDAC Chemistry (01-08)
Center for Terascale Code Design and Development Environments	Jim Kohl (ORNL)	Ames Lab, Georgia Tech, Iowa State, Michigan State, Oregon, ORNL, Virginia Tech	SciDAC ISIC (01-07)
Center for Collaborative Problem Solving in the Earth Sciences Community	Debbie Gracio (PNNL)	ANL, Michigan, NCAR, ORNL, PNNL	SciDAC Collaboratories (01-06)

Title	Lead PI	Participating Institutions	Program
Research on Secure Collaborative Environments for Modeling On-Chip Copper Metallization	R. C. Alkire (Illinois)	Illinois	NSF ITR
Data Parallel Component Software	Steve Parker (Utah)	Utah	NSF ITR
CardioPSE: Computational Bioengineering of the Heart	C. Johnson (Utah)	Indiana, ORNL, Tennessee, Utah	NSF ITR
Integration efforts for NWChem and GAMESS software	Ricky Kendall	Ames Laboratory	N/A
Need for component technology in ASCI program – future collaborations	John Ambrosiano	Los Alamos National Laboratory	N/A
C-SAFE ASCI ASAP Center	David Pershing	University of Utah	N/A
Computational Workbench Environment for Virtual Power Plant Simulations	Michael Bockelie	Reaction Engineering International	Vision 21