# Components, the Common Component Architecture, and the Climate/Weather/Ocean Community

**J. Walter Larson, Boyana Norris, and Everest T. Ong**
*Mathematics and Computer Science Division, Argonne National Laboratory*
*9700 S. Cass Avenue, Argonne, IL 60439*
{larson,norris,eong}@mcs.anl.gov
**David E. Bernholdt, John B. Drake, Wael R. Elwasif, and Michael W. Ham**
*Computer Science and Mathematics Division, Oak Ridge National Laboratory*
*P.O. Box 2008, Oak Ridge, TN 37831-6016*
{bernholdtde,drakejb,elwasifwr,hammw}@ornl.gov
**Craig E. Rasmussen**
*Advanced Computing Laboratory, Los Alamos National Laboratory*
*MS B287, Los Alamos, NM 87545*
rasmussn@lanl.gov
**Gary Kumfert**
*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*
*Box 808, L-365, Livermore, CA 94551*
kumfert1@llnl.gov
**Daniel S. Katz**
*Jet Propulsion Laboratory, California Institute of Technology*
*4800 Oak Grove Drive, Pasadena, CA 91109*
Daniel.S.Katz@jpl.nasa.gov
**Shujia Zhou**
*Northrop Grumman IT/TASC*
*4801 Stonecroft Blvd, Chantilly, VA 20151*
szhou@pop900.gsfc.nasa.gov
**Cecelia DeLuca and Nancy S. Collins**
*Scientific Computing Division, National Center for Atmospheric Research*
*P.O. Box 3000, Boulder, CO 80307*
{cdeluca,nancy}@ucar.edu

## ABSTRACT

Earth system and environmental models present the scientist/programmer with multiple challenges in software design, development, and maintenance, overall system integration, and performance. We describe how work in the industrial sector of software engineering—namely component-based software engineering—can be brought to bear to address issues of software complexity. We explain how commercially developed component solutions are inadequate to address the performance needs of the Earth system modeling community. We describe a component-based approach called the *Common Component Architecture* that has as its goal the creation of a component paradigm that is compatible with the requirements of high-performance computing applications. We outline the relationship and ongoing collaboration between CCA and major Climate/Weather/Ocean community software projects. We present examples of work in progress that uses CCA, and discuss long-term plans for the CCA-climate/weather/ocean collaboration.

# 1 Introduction

Climate/weather/ocean[1] (CWO) applications abound with challenges in high-performance computing, software development, and data management. Forecasters require highly reliable software. Researchers require analysis tools that offer ease of use and flexibility. Modelers desire access to powerful numerical schemes and acceleration of the model development/validation cycle. The existence of millions of lines of legacy Fortran code and the advent of modern programming languages such as C++ and scripting languages such as Python introduce a language barrier to be surmounted. The above desirable characteristics can be summarized as: *quality*, code resulting from a rigorous development process that includes peer review during the requirements, design, and implementation phases, and also is subject to testing; *modularity*, code that is designed to be re-usable in multiple contexts; *flexibility*, code that is easily adapted to meet new user requirements; *extensibility*, a code base that can be expanded with ease to support new applications; *language interoperability*, the ability to connect code modules implemented in different languages; and *performance*, the ability of the application to produce desirable results within a reasonable user timeframe.

We believe the software quality, modularity, flexibility, extensibility, and language interoperability requirements can easily be met through component-based software engineering, an approach that is well established in the commercial software engineering community. We will defer the formal definition of components until Section 2, but for now it is sufficient to say a component is a code entity that serves a well-defined atomic function, or an often-desired combination of atomic functions, that constitutes a fundamental building block that can be used in combination with other components to construct an application. Component-based approaches address the issue of quality through dedicated components that incorporate the best practices/algorithms for a particular function. Components are the natural consequence of a quest for modularity. Large collections of components serving a particular function create flexibility by allowing the user to choose the most appropriate component to solve the problem at hand. Extensibility is achieved by adding new components serving new or previously unforeseen functionalities. General language interoperability is achieved through the use of interface definition languages, which will be discussed in greater detail in section 2.

At this point it is natural to ask why the CWO community has not embraced the component approach. The chief reason is the final desirable characteristic listed above, *performance*. Computational science and in particular CWO applications have a high degree of computational complexity, and performance, as defined by useful metrics such as time-to-solution or throughput has been of paramount importance. A natural consequence of this has been the construction of tightly written legacy codes in which functionalities are sometimes entangled because this approach yields superior performance. The advent of distributed-memory parallel computing paradigms such as

[1] The authors are aware that the computational science and software engineering problems discussed in this paper apply across the whole spectrum of earth system and environmental modeling, and their comments regarding CWO apply to this whole field of endeavor.

MPI[2] have further raised the bar for any different software paradigm to enter the CWO application arena.

In Section 2 we describe component-based software engineering and the Common Component Architecture. In Section 3 we report on ongoing collaborations between CCA and major CWO community projects. In Section 4 we summarize progress made and outline future work.

## 2 Components and the Common Component Architecture

In this section we describe the background of component-based software engineering, describe component approaches originating from the commercial sector, and explain the need for a distinct component approach specifically tailored to high-performance computing.

### 2.1 Component-Based Software Engineering

Component-based software engineering (CBSE) is an emerging approach to help manage the complexity of large-scale software systems and increase the productivity of software developers and users. In CBSE, units of software are encapsulated as "components," which interact with other components only through well-defined interfaces. This approach allows the internal implementation of the component to remain opaque to the rest of the world, and presumably hides much of the complexity of the software. These more manageable units of software can be composed together to form applications; in many cases, well-designed components will be reusable across a number of different applications without internal changes. Components also provide a natural decomposition of applications involving multiple time or length scales, or multiple physical phenomena such as CWO codes.

When many components are available that conform to standardized interfaces for solvers and other such capabilities, the construction of complex applications is greatly simplified, and adaptation to different problems or computational environments can be as easy as swapping one component implementation with another more suitable to the new situation. This provides a "plug-and-play" environment for the construction of applications. Many component environments provide a graphical tool for the assembly of applications, which further simplifies the task of assembling complex applications from components. In addition to the resultant increase in productivity, CBSE also helps researchers focus on application aspects of particular interest and simplifies group software development because components are natural units of work for individuals or small groups and are largely decoupled from other components once their interfaces are defined.

CBSE builds on object-oriented programming concepts but does not require the use of object-oriented languages. It can be viewed as an extension and refinement of the

[2] Message Passing Interface. For more details see http://www.mct.anl.gov/mpi .

common practice of using widely shared software libraries as the basis on which to build an application, offering significant advantages such as easier integration of multiple libraries into a single application and the ability to rigorously enforce the published interfaces for the library. CBSE also has many similarities with the common practice of developing domain-specific computational frameworks that provide an infrastructure to build a variety of applications in a given scientific discipline. We emphasize, however, that CBSE is a more general and flexible approach. One can think of building the domain-specific infrastructure in the form of components, but in the CBSE approach those components could be more easily reused in applications outside the domain as well.

CBSE has most strongly benefited software development on the Internet, in distributed computing, and in business. Several component models are in routine use, such as Enterprise JavaBeans (EJB; Sun Microsystems), Microsoft's COM/DCOM (Microsoft Corp., 1994-2003; Microsoft Corp., 1996-2003), and the Common Object Request Broker Architecture (CORBA; Object Management Group, 2002). Unfortunately, these commodity component architectures suffer from various deficiencies that limit their use in high-performance computing (HPC) contexts. Many component models are designed for situations in which performance criteria are based on human response times on the order of a tenth of a second and cannot provide the performance required in HPC environments, where the relevant time scales are typically measured in nanoseconds. A second major limitation is that commodity component models are designed for distributed computing and have no concept of tightly coupled parallelism appropriate for typical HPC systems. Additional problems concern language support, data types, and platforms that are important to high-performance scientific computing. These issues have limited adoption of CBSE in the HPC community. Indeed, frustration with the situation led to the formation in 1998 of the Common Component Architecture Forum.

## 2.2 The Common Component Architecture

The Common Component Architecture[3] (CCA) is a grass-roots effort to bring the benefits of component-based software engineering to high-performance scientific computing. The CCA is specifically designed to preserve the performance of components on the same processor, to support both tightly coupled parallel and distributed computing, and to simplify the incorporation of existing code into the CCA environment. The CCA uses the Babel[4] (Dahlgren et al., 2003) language interoperability tool to allow components written in various languages (currently Fortran, C, C++, and Python) to interoperate, providing the necessary support for important scientific languages (namely Fortran) and data types. Recognizing the tremendous investment in software that already exists in the HPC world, the CCA is also designed to minimize the burden of incorporating existing software into the component environment.

The primary task of the CCA Forum is to develop the specification for the component architecture, which defines the rights and responsibilities and the relationships between

[3] http://www.cca-forum.org
[4] http://www.llnl.gov/CASC/components/

the various elements of the model. Briefly, these are as follows:

- *Components* are units of software functionality that can be composed together to form applications. Components encapsulate much of the complexity of the software inside a black box and expose only well-defined interfaces to other components.
- *Ports* are interfaces through which components interact. Specifically, CCA ports provide procedural interfaces that can be thought of as a class or an interface in object-oriented languages, or a collection of subroutines, or a module in a language such as Fortran 90. Components may *provide* ports, meaning they implement the functionality expressed in the port, or they may *use* ports, meaning they make calls on that port provided by another component.
- The *framework* holds CCA components as they are assembled into applications and executed. The framework is responsible for connecting uses and provides ports without exposing the components' implementation details. It also provides a small set of standard services, defined by the CCA specification, which are available to all components.

The CCA specification is also the focus of a research and development effort aimed at providing software tools supporting the CCA and understanding how best to utilize and implement CBSE in a high-performance computing environment.

The CCA specification is silent on the issue of programming languages. In many tightly constrained software development situations, especially those that rely extensively on legacy code, one can expect that all components of interest will be written in one or a few languages (i.e., Fortran, or Fortran and C++), and one can implement CCA-compliant components and frameworks in this context, using specialized point-to-point solutions to provide the required language interoperability. In order to gain the full benefits of CBSE, however, it is clearly desirable to be able to connect components regardless of the language in which they were written. Therefore, CCA Forum researchers have developed the Babel language interoperability tool. Babel provides an environment in which all supported languages are treated as peers. Interfaces are expressed in the Scientific Interface Definition Language (SIDL), which is then compiled to produce the "glue" code necessary to call between languages. The performance overhead costs associated with the Babel-generated glue code have been assessed both by Babel developers and developers of the Hypre library, and found to be low (Bernholdt *et al,* 2002; Kohn *et al,* 2001). Babel is not a least common denominator solution; it provides an object-oriented environment even when those features are not native to a supported language. Babel currently supports Fortran 77, Fortran90/95, C, C++, and Python and can be used outside the CCA environment as well.

The CCA specification supports both parallel and distributed computing and facilitates local performance by allowing direct connection of component interfaces. In a direct connection environment, components on a given processor are loaded into a single process, so that they share the same address space, but are loaded into separate namespaces, so that they cannot see each other. In this way, data "owned" by one component can be passed by reference to another component, alleviating the need to copy

data and use a remote method invocation protocol. This is a key performance feature that distinguishes the CCA from commodity component models, which are designed for distributed computing and do not have a concept of direct connection.

The Ccaffeine framework[5] (Allan et al., 2002), used in this work, is one of several developed by CCA researchers[6] (J. McCorquodale et al., 2001; Kumfert, 2003), supporting parallel and distributed computing. Ccaffeine focuses on tightly coupled parallel computing, in which components in the same processor interact through CCA mechanisms (calls on ports), while interprocessor communication among parallel instances of a component are free to use whatever HPC communications layer they prefer—for example, MPI (MPI Forum, 1994), PVM (Geist et al., 1994), and Global Arrays.[7] This is another performance feature of the CCA: it allows software developers to use the parallel programming tools they are most familiar and comfortable with, and it facilitates the incorporation of existing parallel software into the CCA environment. Different components in an application can use different parallel programming libraries.

Ccaffeine supports both single component, multiple data (SCMD) and multiple component, multiple data (MCMD) operation, which are analogous to the noncomponent SPMD and MPMD programming models. In MCMD operation, the processors available to the job are partitioned and different "programs" (groups of components) run on different groups of processors. Hence, it is well suited to coupled simulations, such as climate models including atmospheric, oceanic, land, and other elements, which may naturally prefer to use different numbers of processors.

Another area of CCA research related to the model coupling idea is MxN parallel data redistribution (Allan et al., 2002; Damevski, 2003; Bertrand and Bramley, in preparation; Larson et al., 2001; Keahey et al., 2001). This involves transferring data from a simulation running on M parallel processes to another running on N processes, where in general $M \neq N$, so that the data is not simply transferred but must be reorganized into a new distribution as well. To facilitate model coupling and similar applications, CCA researchers are exploring generalized data redistribution capabilities, which can be implemented as components that are explicitly connected to components needing to transfer data or can be used within the framework to implicitly reorganize method arguments as part of a parallel remote method invocation capability.

The CCA Forum encourages and facilitates the development and publication of both "standardized" interfaces and component implementations of those interfaces. Since interfaces are the means by which components interact, standardization of interfaces facilitates reuse and interoperability of components implemented separately. This kind of standardization usually takes place within a scientific domain and need not imply a formal process. For example, a group of users and developers of solvers or meshing software might agree on a set of interfaces (typically expressed in SIDL) they will conform to. With broad participation from members of the CWO community, the Earth

[5] http://cca-forum.org/ccafe
[6] XCAT framework Web site: http://www.extreme.indiana.edu/xcat/
[7] http://www.emslpnl.gov:2080/docs/global/

System Modeling Framework (ESMF)[8] collaboration , described in section 3.2, is developing a set of standardized interfaces for components Earth system applications. Once interfaces are agreed on, users and developers can adapt their existing software to the new interfaces. If multiple implementations of an interface are available, any of them can be used to satisfy a component's "uses" port for that interface, and one implementation can be swapped for another in a simple plug-and-play fashion. With the encouragement and support of the CCA team, many groups are developing components, often encapsulating popular libraries that will be available for use by others. Our long-term goal is to have a rich "marketplace" of CCA components, allowing many applications to be assembled from a combination of off-the-shelf components, combined with a few specially developed components representing the researcher's particular interests or methods.

# 3 Collaborations between CCA and the CWO Community

In this section we describe two major community efforts to modernize and improve the performance of the software used in CWO applications, namely the DOE Climate Change Prediction Program and the NASA-funded interagency Earth System Modeling Framework project, and the CCA outreach to both of these projects.

## 3.1 DOE Climate Change Prediction Program

The Department of Energy's Climate Change Prediction Program (CCPP) is funded under the DOE Scientific Discovery through Advanced Computing Program (SciDAC) initiative. One major initiative under CCPP is an interagency collaboration with NSF's National Center for Atmospheric Research (NCAR) to improve the performance, software quality, and scientific content of the Community Climate System Model[9] (CCSM).

CCSM is a coupled climate model, comprising several mutually interacting component models—atmosphere, ocean, sea ice, land-surface, river routing, and a flux coupler. The flux coupler provides the following services to the coupled model: intermodel state and flux data transfer, interpolation between model grids, calculation of fluxes and other diagnostic quantities, time averaging and accumulation of flux and state data, merging of flux and state data from multiple model sources for use by another model, and overall coordination of the coupled model's execution. Our collaboration with CCSM is focused on two areas. We are prototyping use of CCA at the system integration level, the model subcomponent level, and the algorithmic level. We are also exploring the use of CCA to package portions of the Model Coupling Toolkit—the foundation code on which CCSM's flux coupler is built—as CCA components. Since CCSM plans to adopt the standard interfaces being developed by the ESMF project (see section 3.2), the CCPP work is being performed in collaboration with the ESMF group.

[8] http://www.esmf.ucar.edu
[9] http://www.ccsm.ucar.edu

### 3.1.1 Model Coupling Toolkit

The current CCSM flux coupler was implemented by using the Model Coupling Toolkit[10] (MCT). MCT (Larson *et al.*, 2001; Ong *et al.*, 2002) is a software package for constructing parallel couplings between MPI-based distributed-memory parallel applications. MCT provides the following often-used objects and services needed to construct distributed-memory parallel coupled models:

- A lightweight component model registry;
- A multifield data storage object;
- Domain decomposition descriptors, communications schedulers for intercomponent parallel data transfer and intracomponent parallel data redistribution, and the facilities to implement intercomponent handshaking;
- A class encapsulating distributed sparse matrix elements and communication schedulers used in performing interpolation as parallel sparse matrix-vector multiplication;
- A data object for describing physical grids capable of supporting grids of arbitrary dimension and unstructured grids that also support masking of grid elements;
- Spatial integral and averaging facilities that include paired integrals and averages for use in conservation of global flux integrals in intergrid interpolation;
- Registers for time averaging and accumulation of field data; and
- A facility for merging of state and flux data from multiple sources for use by a particular model

MCT supports both sequential and concurrent couplings and can support multiple executable images if the implementation of mpirun used supports this feature. MCT is implemented in Fortran90, and its programming model is scientific-programmer-friendly, based on F90 module use to declare MCT-type variables and invocation of MCT routines to create couplings.

The first major focus of the MCT-CCA collaboration has been the creation of a migration path between MCT-based coupling and a CCA-compliant component-based approach (Larson *et al.*, 2003). This is being accomplished by using MCT to create an implementation of the ESMF standard component-level interfaces (another implementation of the ESMF interfaces is being developed at NCAR). The ESMF data types and programming model, described in more detail in section 3.2, offer a bridge between the lower level functionality provided by MCT and the generic component interfaces provided by CCA. The `ESMF_State` type encapsulates the data associated with a model's state, specifically the field data, the physical grid on which they reside, and the data's domain decomposition. The `ESMF_GridComp` type encapsulates all of a model's input and output states (each of which is associated with and individual `ESMF_State` instance) and other useful information needed for coupling the model to the outside world. The `ESMF_CplComp` type is used to connect an output `ESMF_State` from a source model to an input `ESMF_State` of another model. The

[10] http://www.mcs.anl.gov/mct

basic unit of software to be cast as a component is any CCSM component model that is currently coupled using MCT. Each component is implemented by using the Fortran derived type `ESMF_GridComp`, which sets the standard for the component's coupling interfaces. In this approach, component composition can be accomplished by using the following methods: composition and static linkage using Fortran, construction of C++ wrappers and composition using Ccaffeine—the so-called "CCA Classic" approach, and description of component interfaces using SIDL and Babel and composition using a CCA-compliant framework. Currently, this component approach supports SCMD component scheduling and coupling only, but it will soon be expanded to support MCMD coupling as well.

We have begun the task of describing ESMF-compliant MCT interfaces using SIDL. The first and obvious motivating factor is the goal of repackaging MCT as a collection of CCA-compliant components. Prospective MCT-based components include distributed multi-field data storage, parallel sparse matrix-vector interpolation, global integrals, and averages. We present in Section 3.1.3 an example of how these components might be applied. Another application of the SIDL interface description is the extension of MCT's programming model to languages other than Fortran90. This is an example of how the Babel interlanguage interoperability tool can be used in a context outside of CCA.

### 3.1.2 CCSM System Integration

At the system integration level, we are prototyping the use of CCA to cast the CCSM's component models as CCA-compliant components. We have developed skeleton components for the atmosphere, ocean, sea ice, land surface, river routing, and flux coupler. These components can then be instantiated and connected by using the Ccaffeine GUI (Figure 1); their subsequent execution is similar to the ESMF-CCA prototype described in Section 3.2.2. These skeleton components are being expanded to have the capabilities of the CCSM "data" models that are used to perform system integration tests on the CCSM coupler. The components will then entail sufficient computational complexity that we can begin to assess in the coupling context any performance costs associated with CCA components versus the current coupler.

### 3.1.3 Community Atmosphere Model

An example of a CCA application targeted at the model subcomponent level is the refactoring of CCSM's atmosphere—the Community Atmosphere Model[11] (CAM). CAM has been refactored to disentangle its dynamical core from its subgridscale physics parameterizations. This split allowed a proliferation of dynamical cores, and CAM currently has three: a pseudo-spectral method, the Rasch-Williamson semi-Lagrangian method, and the finite-volume Lin-Rood scheme. Standardization of the interfaces to these dynamical cores and the overall subgridscale physics package is under way. Once complete, these software modules can be packaged as CCA components. This will allow one to use CCA to compose and invoke a stand-alone version of CAM in which the user

[11] http://www.ccsm.ucar.edu/models/atm-cam/

may plug in the dynamical core of choice. Furthermore, the developer of a new dynamical core will need only code it to the interface standard, and it will be easily integrated into the model for validation.

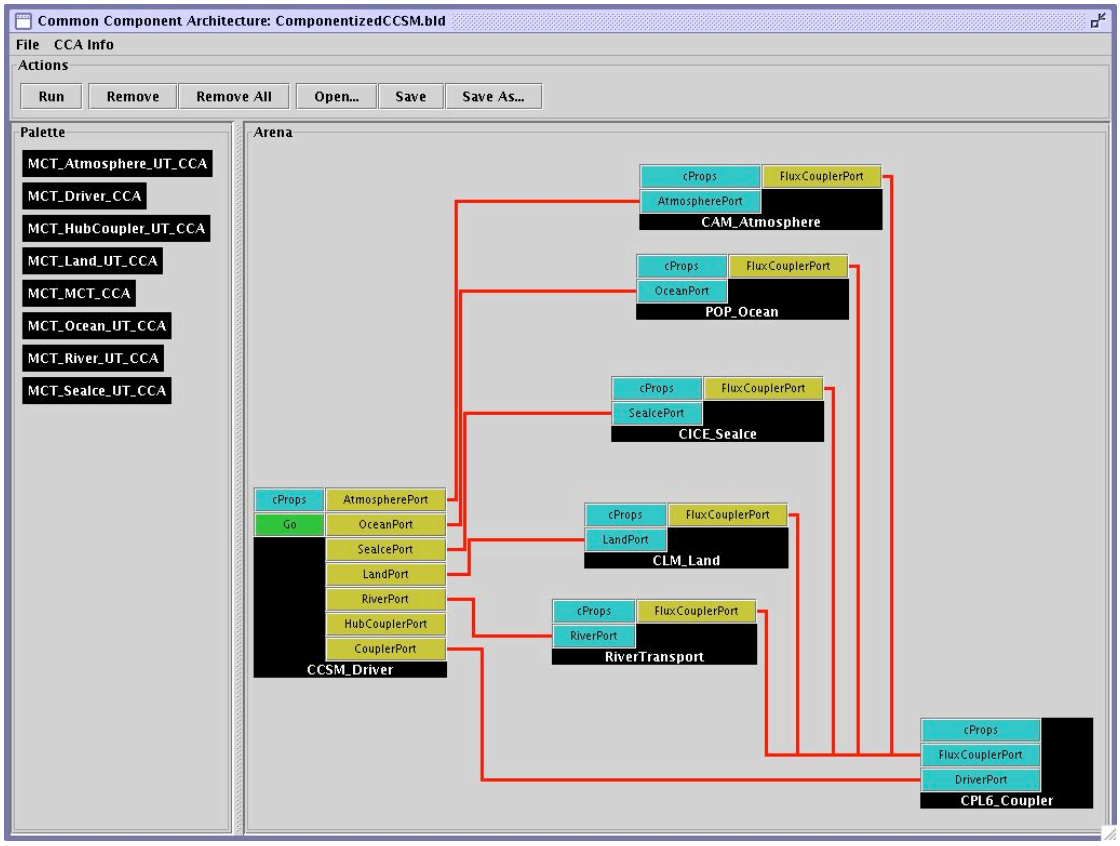


**Figure 1. Prototype of CCA component-based system integration of the CCSM.**

### 3.1.4 River Transport Model

The river transport model (RTM) in the CCSM version 2 computes river flow on a 0.5° latitude-longitude grid and uses no parallelism in its calculations. Future science goals for the RTM include support for significantly higher-resolution (up to 100x the number of catchments) and catchment-based unstructured grids, support for transport of dissolved chemical tracers, and inclusion of the effects of water storage in the form of reservoirs. At the current resolution, the lack of parallelism in the RTM calculation does not retard the overall execution of CCSM, but the process of coupling to the RTM, which requires gathering (scattering) data to (from) a single processor when sending (receiving) data to (from) the RTM, does impose a bottleneck. This coupling communications cost exceeds the computation cost of the RTM.

[12] http://www.ccsm.ucar.edu/models/atm-cam/

To incorporate distributed-memory parallelism in the RTM and meet its science goals, we are developing a completely new implementation of the RTM using MCT. In this approach, we view the problem of river transport as a directed graph with the nodes representing catchments and the directed edges representing the dominant inter-catchment flow paths. The river transport calculation is then implemented as a parallel sparse matrix-vector multiply, and the load balance for this operation can be computed by using graph partitioning. Mass conservation is ensured through the calculation of global mass integrals before and after the matrix-vector multiply. This problem is easily solved using MCT. The ESMF-compliant MCT multifield data storage object is used to store runoff and tracer concentration data; MCT's domain decomposition descriptors are used to describe the load balance; MCT's parallel sparse matrix-vector multiply is used to perform the water and chemical tracer transport; and MCT's paired global spatial integral facility is used to carry out the mass balance and mass conservation calculations.

The component-based approach used for key MCT capabilities makes the path to a component-based RTM clear. Four MCT-based components will be used to implement the RTM: a distributed multifield array used to store water and chemical tracer concentrations; a parallel sparse-matrix vector multiply, which will be used to compute water and chemical tracer transport; a component describing the physical grid (i.e., the locations and sizes of the catchments); and a paired spatial integral component, which will be used to enforce mass conservation.

## 3.2 The Earth System Modeling Framework

The Earth System Modeling Framework (ESMF) is a national effort to develop common software utilities ("ESMF infrastructure") and coupling services ("ESMF superstructure") in the climate, weather, and data assimilation domains. The objectives of the ESMF are to facilitate:

- an extensible, hierarchical achitecture for structuring complex CWO models;
- interoperability of modeling components among major centers;
- lower cost of entry into Earth system modeling for university and smaller groups;
- sharing of common software utilities and services.

Advances in Earth system simulation and prediction require the incorporation and evaluation of new physical processes such as biogeochemistry and atmospheric chemistry, which can be represented as independent software components. Standardization of interfaces for new and existing components in CWO applications via ESMF and CCA creates a clear path for researchers throughout the community to develop software that can be utilized in multiple applications and contexts. It also allows researchers to draw from a broad spectrum of available components when developing new applications. The functionality and collaborative potential that the ESMF represents are needed to meet forthcoming scientific challenges.

The 15 application testbeds for the ESMF include the CCSM mentioned in the previous section of this paper, the Weather Research and Forecast[14] (WRF) model, models from the NOAA Geophysical Fluid Dynamics Laboratory, the National Centers for Environmental Prediction, MIT, and the NASA Global Modeling and Assimilation Office. Each of these applications either *is* a multi-component, coupled modeling system or *has the potential* to be used in such a modeling system. As collaborators on the ESMF project, these groups will use a standard `ESMF_GridComp` data type to represent their atmosphere, ocean, land, sea-ice and data assimilation components, and an `ESMF_CplComp` coupler data type, built on underlying toolkits for regridding and communication, to connect gridded components into applications.

Like CCA, the ESMF provides a generic component abstraction that connects user code components to the framework through a `SetServices()` method. Unlike CCA, ESMF customizes methods and data structures for Earth system models, and provides utilities and services specific to the CWO domain. For example, the data exchanged between ESMF components is packed into a data structure called an `ESMF_State`, which is a container for data types that represent geophysical fields. Since ESMF applications typically perform a setup, timestep through numerical solution of PDEs, and then shut down, ESMF codes are required to have `Initialize()`, `Run()`, and `Finalize()` methods.

The ESMF and CCA groups have worked closely together to ensure interoperability between the ESMF and CCA frameworks; i.e., to guarantee that ESMF components may be used within CCA, and that ESMF will be able to utilize the wide variety of CCA-compliant components that exist or are under development. An ESMF-CCA prototype, described in more detail in the next section, has been developed that uses the component registration methodology and GUI from CCA. ESMF components do not need to be modified, but can simply be wrapped to become CCA-compatible. The prototype demonstrates that interoperability between CCA and the ESMF is entirely plausible and in fact not that onerous, since it is facilitated by the similar architectural principles that underlie both frameworks. The capacity for interoperability between these frameworks is exciting, since it opens the possibilities of bringing many Earth system modeling components into the CCA arena, and making CCA tools and services available to a host of Earth system applications.

### 3.2.1 ESMF-CCA Interoperability

Several notable distinctions exist between CCA and the ESMF component frameworks. The primary distinction is that the CCA provides a general component model, while the ESMF provides a specialized component model tailored to a specific application domain (CWO applications). Moreover, the CCA enables dynamic composition of component-based applications at run time, while the ESMF dictates static linkage of a component hierarchy. The CCA does not provide concrete classes with which to build components, while the ESMF does (the ESMF infrastructure). In this sense, the CCA is *component*

[14] http://wrf-model.org

framework, while the ESMF is an *application* framework.

These differences suggest the intriguing possibility of using the CCA component model and the ESMF application framework together to build CWO applications. This would bring the dynamism of the CCA component model to CWO developers. It would also enable coupling to CCA components from within the ESMF framework, allowing ESMF components to step outside of bounds of the ESMF_Initialize(), ESMF_Run(), ESMF_Finalize() paradigm if a richer component interface is required. Furthermore, the union of these two frameworks would provide the CCA community with access to ESMF components.

The key question is: How do we build a bridge linking the two component models? Fortunately, the ESMF framework provides hooks to easily accomplish this task. ESMF components must provide a special function to register their interface functions with the ESMF framework. Thus, a CCA component (masquerading as an ESMF component) need only provide this special function as one of its CCA ports.

The joint ESMF-CCA component architecture is fairly straightforward. ESMF components are wrapped with a thin layer of C++ code to provide the CCA component veneer. This wrapper allows an ESMF-CCA application to be composed by selecting from a palette of ESMF-CCA components, using the Ccaffeine CCA framework. A special version of the ESMF component registration function is provided for all ESMF-CCA components, to allow them to register their ESMF interface functions (initialize, run, finalize). Once all components are created and connected, the CCA framework passes control flow over to the ESMF framework.

A major advantage of this dynamic approach is that components can be easily substituted for one another without modifying user code (as long as the swapped components conform to the same interface). As it now stands, one must modify the ESMF superstructure to replace a component, as the complete component hierarchy is coded into an ESMF application.

We recognize that climate and weather models do not typically require run-time swapping of geophysical components, since it takes on the order of months to tune and validate these applications, and months to run long climate simulations. However, run-time swapping may be useful for communication and IO components.

### 3.2.2 ESMF Prototype Based on CCA

A key difference between the CCA component environment and the ESMF component environment is ESMF's requirement for the user to supply additional standard methods (beyond those required for registering the component in the framework). Our ESMF-CCA Prototype uses CCA's Uses-Provides design pattern to couple components. In addition to having an interface method for component registration, the ESMF-CCA components also have methods similar to ESMF standard component Initialize(), Run(), and Finalize() methods. The data exchange between ESMF-CCA components is through

a standardized, self-describing data type similar to the ESMF_State. We are working on a C++ version of the ESMF component interface so that ESMF components can be used with CCA in a very straightforward way (Zhou *et al,* 2003). In the following, we illustrate the ESMF-CCA prototype through a typical sequential coupling between an atmosphere model and an ocean model. At the beginning of a simulation, five components are created: a driver (Climate), the atmosphere (Atm), the ocean (Ocn), a coupler from atmosphere to ocean (CplAtmXOcn), and a coupler from ocean to atmosphere (CpleOcnXAtm). In the ESMF-CCA prototype, we use the CCA setServices() utility to register these five components. In addition, data exchange is implemented through import or export states similar to the ESMF_State datatype. The component execution sequence of events is as follows: (1) The atmosphere component *provides* its data at its boundary, exportAtm, to the coupler, CplAtmXOcn. (2) CpleAtmXOcn *uses* exportAtm, transforms it into importOcn with interpolation routines, and then *provides* importOcn to the ocean component. (3) With importOcn, the ocean component runs its solver for the evolution equations and then *provides* its data at the boundary, exportOcn, to the coupler, CplOcnXAtm. (4) The coupler, CpleOcnXAtm, *uses* exportOcn, transforms it into importAtm, and *provides* importAtm to the atmosphere component. (5) The atmosphere component *uses* importAtm, runs its solver for evolution equations, and then *provides* its data at the boundary, exportAtm, to the coupler, CpleAtmXOcn. The coupling sequence can be well represented in a screenshot of the Ccaffeine GUI used in the simulation (Figure 2).

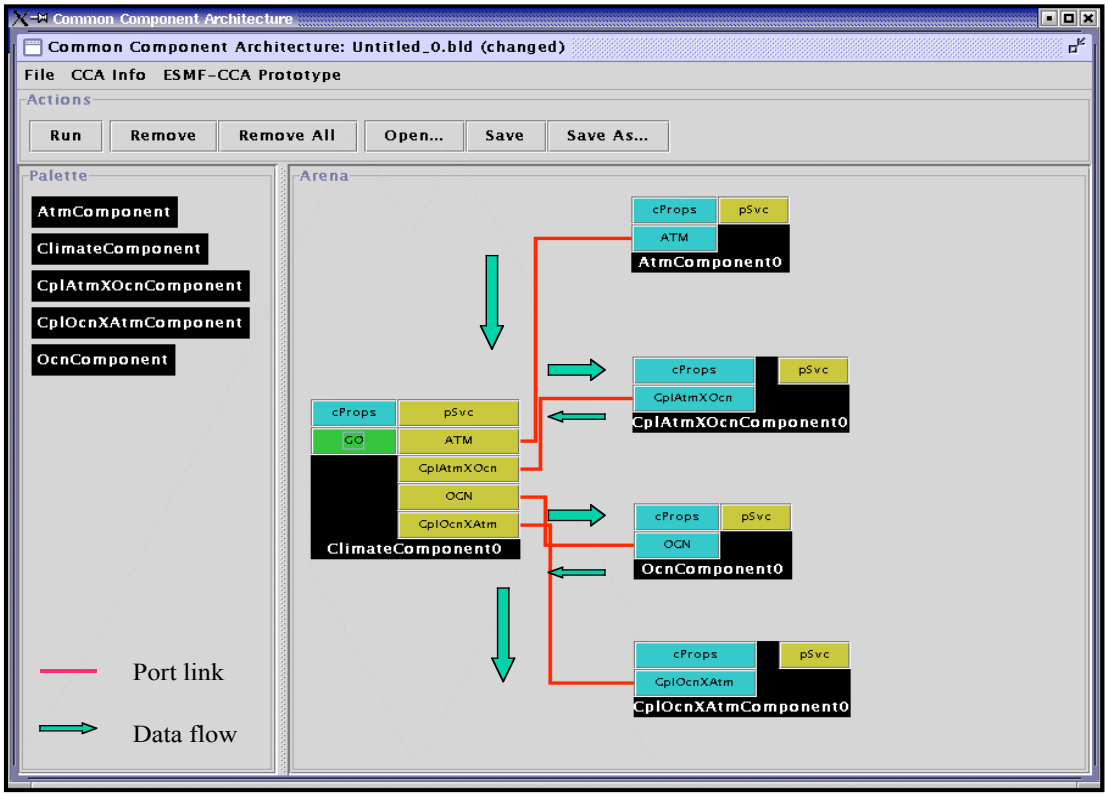


**Figure 2.** CCA wiring diagram showing component relationship in a simulation of coupled atmosphere and ocean model components.

### 3.3 UCLA Coupled Climate Model and Distributed Data Broker

An ongoing project at JPL consists of a demonstration of CCA with three specific climate components: the UCLA Atmospheric General Circulation Model (AGCM; Wehner et al., 1995), the LANL POP Ocean General Circulation Model (OGCM; Smith et al., 1992), and the UC Berkeley/UCLA Distributed Data Broker (DDB; Drummond et al., 2001). We will not discuss the AGCM or OGCM in this paper; they have been discussed elsewhere and are fairly well known to the climate community. Since the DDB is not as well known, however, we include here a short description.

The DDB is used in two phases. First, components that will use it to exchange data register with it by signing up to produce or to consume data. In this phase, they also describe the global view of the data they have or want, as well as the mapping of this data onto the processors. The DDB then calculates what messages will have to be sent to transfer data from one component that is producing it to another component that is consuming it. This calculation is done on a single processor at the end of this first phase, but the information about the messages themselves is stored on the processors that will be sending or receiving them. This is why the DDB is called distributed. In the second phase, the actual messages are sent when the components on a given processor signal that they have produced or are ready to consume data. Also, any needed interpolation is done by the DDB, if the grids are not coincident or if no data exists at some number of points.

The CCA version of this application is expected to be fairly straightforward. We note that since the work is being done with the Ccaffeine framework (Allan et al., 2002), which does not currently have the ability to instantiate one component on a subset of processors (AGCM), another on the remaining processors (OGCM), and a third on all the processors (DDB), we use a version of the coupled application where the OGCM has been turned into a library that is called from the AGCM driver code on a subset of the AGCM nodes. Hence, from the viewpoint of Ccaffeine, the AGCM is a driver component (having only a *go* port and multiple *uses* ports), and the DDB is a server component (having only *provides* ports.) Additionally, we think that the DDB component will be useful in other applications, both in climate and other fields.

## 4 Conclusions and Future Work

We have described in detail the Scylla of software complexity and Charybdis of performance degradation that faces the CWO scientist/programmer. We have described in detail how industrial software engineers have created the CBSE paradigm to tame software complexity in their applications. We described how commercial CBSE solutions have human reaction time as their typical intercomponent latency and how this timescale is inadequate to the support high-performance computing applications endemic to computational science and engineering, and in particular the CWO community. We have described a component approach that has as its chief goal imposing low overhead in order to support scientific simulation—the Common Component Architecture. We presented in detail CCA's outreach to the CWO community, and the early promising results of this collaboration.

The work presented in this paper varies in maturity from conceptual design to prototype to working CCA-based application. We feel the current work represents a bridgehead into the CWO community that can be expanded substantially by demonstrating how CCA's component approach will position CWO applications better to utilize the wide variety of scientific computing components presently under development. We believe components of particular interest are: ODE system solvers for use in atmospheric chemistry and global biogeochemistry; spatial mesh generators and PDE solvers for three-dimensional fluid flow for use in atmosphere, ocean, and sea ice dynamical cores; and parallel i/o components. We recognize that many of these components represent leading edge technology, and that this may be hard to integrate in operational forecast systems that represent trailing edge technology[15], and the process of adoption by the CWO community may proceed slowly. We believe that the benefits afforded by CCA will cause this pace to accelerate as time progresses.

**Acknowledgments**

This work has been supported in part by the U. S. Dept. of Energy's Scientific Discovery through Advanced Computing initiative, through the Center for Component Technology for Terascale Simulation Software, of which ANL, ORNL, LANL, and LLNL are members. Some of the authors at ANL and ORNL receive partial support from the SciDAC Climate Change Prediction Program's Collaborative Design and Development of the CCSM for Terascale Computers. Some of the authors at NCAR, JPL, Northrop Grumman, and ANL receive either full or partial support from the Earth System Modeling Framework project, which is supported by NASA's Computation Technologies (CT) Project, part of the Earth Science Technology Office (ESTO), under a contract with the National Aeronautics and Space Administration.

Some of the work reported in this paper was performed during two CCA-sponsored climate applications coding sessions (a.k.a. "Climate Camps"), and we are grateful to the institutions that hosted these events. The CCA Climate Camp took place in May 2003 and was hosted by Rob Armstrong at Sandia National Laboratories. The second CCA Climate Camp took place in September 2003 and was hosted by Randall Bramley of the Computer Science Department at Indiana University. During these coding sessions, the authors enjoyed and gratefully acknowledge considerable help from their hosts and Ben Allan of Sandia National Laboratory, and Tom Epperly, and Tammy Dahlgren of Lawrence Livermore National Laboratory.

Argonne National Laboratory is managed by the University of Chicago for the U.S. Department of Energy under contract W-31-109-ENG-38. Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the US Department of Energy under contract DE-AC-05-00OR22725. Lawrence Livermore National Laboratory and Los Alamos National Laboratory are managed on behalf of the US Department of Energy by the University of California. Some of the work in this paper was carried out by Northrop

[15] Here the term "trailing edge" is not meant to be pejorative, but instead meant to reflect the high degree of criticality and requirement for ultra-reliability present in these systems.

Grumman/TASC with funding provided by NASA's Computation Technologies (CT) Project, part of the Earth Science Technology Office (ESTO), under a contract with the National Aeronautics and Space Administration. Some of the work in this paper was carried out by the Jet Propulsion Laboratory at the California Institute of Technology under a contract with the National Aeronautics and Space Administration. The National Center for Atmospheric Research is managed for the National Science Foundation for the University Consortium for Atmospheric Research.

**References**

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," *Concurrency and Computation*, **14**, 1–23 (2002).

David E. Bernholdt, Wael R. Elwasif, James A. Kohl, and Thomas G. W. Epperly, "A Component Architecture for High-Performance Computing," in *Proceedings of the Workshop on Performance Optimization via High-Level Languages and Libraries (POHLL-02)*, New York, NY. 22 June 2002

F. Bertrand and R. Bramley, "A Distributed CCA Framework Based on MPI," manuscript in preparation.

T. Dahlgren, T. Epperly, and G. Kumfert, "Babel Users' Guide, version 0.8.4," CASC Technical Publication, Lawrence Livermore National Laboratory (2003).

K. Damevski, "Parallel RMI and M-by-N Data Redistribution using an IDL Compiler," M.S. thesis, University of Utah (2003).

L. A. Drummond, J. Demmel, C. R. Mechoso, H. Robinson, K. Sklower, and J. A. Spahr, "A Data Broker for Distributed Computing Environments," *Proceedings of the International Conference on Computational Science (ICCS) 2001*, edited by V. N. Alexandrov, J. J. Dongarra, B. A. Juliano, R.S. Renner, and C. J. K. Tan, *Lecture Notes in Computer Science,* vol. 2073, Springer-Verlag, Berlin, pp. 31–40 (2001).

G. A. Geist, A. Gebuelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam,*PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA (1994).

C. Hill, C. Deluca, V. Balaji, M. Suarez, A. DaSilva, J. Anderson, B. Boville, C. Cruz, N. Collins, T. Craig, R. Hallberg, M. Iredell, R. Jacob, P. Jones, B. Kauffman, E. Kluzek, J. Larson, J. Michalakes, D. Neckels, W. Sawyer, E. Schwab, S. Smithline, A. Tryanov, W. Yang, M. Young, and L. Zaslavsky, "The Architecture of the Earth System Modeling Framework," accepted for publication, *Computing in Science and Engineering* (2003).

K. Keahey, P. Fasel, and S. Mniszewski, "PAWS: Collective Interactions and Data Transfers," in *Proceedings of the High Performance Distributed Computing Conference*, San Francisco, CA, August 2001.

Scott Kohn, Gary Kumfert, Jeff Painter, and Cal Ribbens, "Divorcing Language Dependencies from a Scientific Software Library," *Tenth SIAM Conference on Parallel Processing*, Portsmouth, VA, March 12-14, 2001.

G. Kumfert, "Understanding the CCA Specification Using Decaf," Lawrence Livermore National laboratory UCRL-MA-145991 (2003). Available on-line at http://www.llnl.gov/CASC/components/docs.html .

J.W. Larson, R.L. Jacob, I.T. Foster, and J. Guo, "The Model Coupling Toolkit," in *Proceedings of the International Conference on Computational Science (ICCS)* 2001, edited by V. N. Alexandrov, J. J. Dongarra, B. A. Juliano, R. S. Renner, and C. J. K. Tan, *Lecture Notes in Computer Science,* vol. 2073, Springer-Verlag, Berlin, pp. 185–194 (2001).

J. W. Larson, C. E. Rasmussen, O. Volberg, E. T. Ong, and R. L. Jacob, "From Large Coupled Model to a Component-based Application: Building Bridges between the Model Coupling Toolkit, the Earth System Modeling Framework, and the Common Component Architecture," in preparation, 2003.

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

Microsoft Corporation, "Component Object Model Specification," available on-line at http://www.microsoft.com/com/resources/comdocs.asp (1994–2003).

Microsoft Corporation, "Distributed Component Object Model Specification," available on-line at http://www.microsoft.com/com/tech/dcom.asp (1996–2003).

Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," *Intl. J. of Supercomputer Appls. and High Perf. Comp.* **8**, 159–416 (1994).

J. Nieplocha, R. Harrison, and R. J. Littlefield, "Global Arrays: A Non-Uniform-Memory-Access Programming Model for High-Performance Computers," *J. Supercomputing* **10**, 169 (1996).

Object Management Group, "CORBA Component Model," available on-line at http://www.omg.org/technology/documents/formal/components.htm (2002).

E. Ong, J. Larson, and R. Jacob, "A Real Application of the Model Coupling Toolkit", in Proceedings of the 2002 International Conference on Computational Science, edited by C. J. K. Tan, J. J. Dongarra, A. G. Hoekstra, and P. M. A. Sloot, *Lecture Notes in Computer Science*, vol. 2330, Springer-Verlag, Berlin, pp. 748–757 (2002).

R. D. Smith, J. K. Dukowicz, and R. C. Malone, "Parallel Ocean General Circulation Modeling," *Physica D*, **60**, 38 (1992).

Sun Microsystems, "Enterprise JavaBeans Downloads and Specifications," available on-line at http://java.sun.com/products/ejb/docs.html (1998–2003).

M. F. Wehner, A. A. Mirin, P. G. Eltgroth, W. P. Dannevik, C. R. Mechoso, J. Farrara and J. A. Spahr, "Performance of a Distributed Memory Finite-Difference Atmospheric General Circulation Model," *J. Parallel Comp.*, **21**, 1655–1675 (1995).

S. Zhou, A. da Silva, B. Womack, and G. Higgins, "Prototyping the ESMF Using DOE's CCA," NASA Earth Science Technology Conference 2003, College Park, MD, June 24–26, 2003, available on-line at the URL http://esto.nasa.gov/conferences/estc2003/papers/A4P3(Zhou).pdf .