NASA/TM—2001–209992

Software Engineering Support of the Third Round of Scientific Grand Challenge Investigations

Earth System Modeling Software Framework Survey
Task 4 Report

Prepared by
*B. Talbot, S. Zhou and G. Higgins**
Northrup-Grumman Information Technology/TASC
4801 Stonecroft Blvd.
Chantilly, VA 20151-3822

G. Higgins, Program Manager

*Corresponding author G. Higgins: (703) 633-8300 x4049; ghiggins@northropgrumman.com

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

May 2002

NASA/TM—2001–209992

Software Engineering Support of the Third Round of Scientific Grand Challenge Investigations

Earth System Modeling Software Framework Survey
Task 4 Report

Prepared by
*B. Talbot, S. Zhou and G. Higgins**
Northrup-Grumman Information Technology/TASC
4801 Stonecroft Blvd.
Chantilly, VA 20151-3822

G. Higgins, Program Manager

*Corresponding author G. Higgins: (703) 633-8300 x4049; ghiggins@northropgrumman.com

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

May 2002

ABSTRACT

One of the most significant challenges in large-scale climate modeling, as well as in high-performance computing in other scientific fields, is that of effectively integrating many software models from multiple contributors. A software framework facilitates the integration task, both in the development and runtime stages of the simulation. Effective software frameworks reduce the programming burden for the investigators, freeing them to focus more on the science and less on the parallel communication implementation, while maintaining high performance across numerous supercomputer and workstation architectures.

This document surveys numerous software frameworks for potential use in Earth science modeling. Several frameworks are evaluated in depth, including Parallel Object-Oriented Methods and Applications (POOMA), Cactus (from the relativistic physics community), Overture, Goddard Earth Modeling System (GEMS), the National Center for Atmospheric Research Flux Coupler, and UCLA/UCB Distributed Data Broker (DDB). Frameworks evaluated in less detail include ROOT, Parallel Application Workspace (PAWS), and Advanced Large-Scale Integrated Computational Environment (ALICE). A host of other frameworks and related tools are referenced in this context. The frameworks are evaluated individually and also compared with each other.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

This report surveys existing modeling frameworks to provide insight into available software that could potentially support the development of a community Earth System Modeling Framework (ESMF). The survey focused on the six frameworks listed below; however, many more were examined in less detail.

- Parallel Object-Oriented Methods and Applications (POOMA)
- Cactus
- Overture (Object-Oriented Tools for Solving CFD and Combustion Problems in Complex Moving Geometry)
- Goddard Earth Modeling System (GEMS)
- National Center for Atmospheric Research (NCAR) Flux Coupler
- University of California, Los Angeles, (UCLA) Distributed Data Broker (DDB)

A Web site has been developed [1] to provide reference to the investigated frameworks, as well as others that are not mentioned in this report. In addition, an interim briefing of the survey results is provided at that site.

This report is a focused rather than a general survey in that it reviews the frameworks in the context of ESMF needs rather than in the context of the specific problem the framework was originally designed to solve. The ESMF needs are expressed in both the Cooperative Agreement Notice (CAN) [2] and in documents arising from meetings of the Common Modeling Infrastructure working group (CMIWG) [3]. These needs are informally documented here. Framework deficiencies relative to ESMF needs should not be considered general weaknesses, since none of the frameworks were specifically developed to address those needs.

The report is divided into three major sections: Introduction, Framework Survey, and Discussions and Recommendations.

Section 2, Framework Survey, which follows this Introduction, is divided into eight major subsections. Subsection 2.1 provides the information collection approach and the major points-of-contact (POC's) within the organizations that developed the framework. Subsections 2.2 through 2.7 provide survey results for the six primary frameworks (listed above). Each subsection is further divided into five sections: introduction, synopsis, description, evaluation, and references. Separate references lists are provided for each framework to simplify the search for the reader and because there is little overlap. This approach is used throughout the document. Subsection 2.8 provides less detailed information on three other frameworks—ROOT, Parallel Application Workspace (PAWS), and Advanced Large-Scale Integrated Computational Environment (ALICE), that may be of interest to the climate community but which were not reviewed at the same level of detail as the others. As indicated earlier, an extensive set of links to these and other frameworks identified in the survey may be found at the survey web site [1].

Section 3, Discussion and Recommendations, is divided in four subsections. Subsection 3.1 is entitled "Motivation for ESMF." This subsection presents results of reviewing the CAN and findings of CMIWG to identify the underlying requirements of a community ESMF. Subsection 3.2 looks at the relative capabilities of the surveyed frameworks relative to ESMF requirements. Section 3.3 provides recommendations on the types of framework solutions that may be possible to satisfy ESMF needs. Finally, Section 3.4 provides the references for all of Section 3.

1.1 References

[1] Survey results. URL: http://esdcd.gsfc.nasa.gov/ESS/esmf_tasc/index.html

[2] Staff, National Aeronautics and Space Administration (NASA), 2000: NASA High Performance Computing and Communications (HPCC)/Earth and Space Sciences (ESS) Cooperative Agreement Notice (CAN) for Solicitation of Round-3 Grand Challenge Investigations: Increasing Interoperability and Performance of Grand Challenge Applications in the Earth, Space, Life, and Microgravity Sciences. URL: http://earth.nasa.gov/nra/current/can00ocs01/

[3] CMIWG. http://janus.gsfc.nasa.gov/~mkistler/infra/master.html

2.0 FRAMEWORK SURVEY

2.1 Information Collection

A variety of sources was used to obtain information about each framework. These sources included Web information, technical documentation (e.g., technical reports, software documents), software source code review, and personal communication with the framework developers. The personal communications were particularly useful and most organizations were gracious enough to provide more than one telephone interview to clarify findings. In the early stages of the survey, a list of basic inquiry areas was formulated to focus the information search. This was done to ensure that similar information was collected from each organization, although it did not necessarily limit the framework review. These inquiry areas included the following:

- **Requirements/capabilities**—Major goals of the framework, current capabilities, parallel implementation, unique capabilities, future capabilities (i.e., are important new features planned?).
- **Language**—Implementation language of the framework and language requirements of applications using the framework.
- **High-level design**—Highest level design in terms of a hierarchical structure diagram or functional diagram.
- **Tools and utilities included**—Libraries, key Fortran subroutines, etc., that might be of interest to the climate community (e.g., grid utilities, parallel communications utilities).
- **Application interface**—How would an application interface with the framework?
- **Other issues**—Maturity of framework (number of users, number of versions issued, number of developers on team), performance optimization (optimization efforts by developers to improve performance), etc.

The points-of-contact (POC's) for the six frameworks are shown in table 1.

Table 1. POC's for the Six Surveyed Frameworks

Framework	Organization	Primary POC's
POOMA	Los Alamos National Laboratory	Julian Cummings
Cactus	Collaborative effort within the relativistic physics community	Ed Siedel
Overture	Lawrence Livermore National Laboratory	David Brown, Bill Henshaw, Dan Quinlan
GEMS	NASA NSIPP and DAO	Max Suarez
Flux Coupler	NCAR	Brian Kauffman and Tom Bettge
Distributed Data Broke	UCLA Atmospheric Sciences Department	Tony Drummond

The results of the survey are provided in the remaining subsections. Subsection 2.2 describes POOMA, 2.3 describes Cactus, 2.4 describes Overture, 2.5 describes GEMS, 2.6 describes NCAR Flux Coupler, 2.7 describes the UCLA Distributed Data Broker, and 2.8 describes ROOT, PAWS and ALICE.

2.2 POOMA

2.2.1 Introduction

Parallel Object-Oriented Methods and Applications (POOMA) originates in the Los Alamos National Laboratory (LANL). It is an object-oriented framework for applications in computational science requiring high-performance parallel computers [1]. This description is based upon POOMA documentation [1, 2], teleconferences with POOMA developers, and the assessment of POOMA by the National Energy Research Scientific Computing Center (NERSC) [3].

2.2.2 Synopsis

Prominent information about POOMA includes the following:

1. **Community of origin**—POOMA originates in the LANL.
2. **Description**—POOMA is a library of C++ classes designed to represent common abstractions in high-performance computing applications.
3. **Team**—The POC is Julian Cummings, who is now at the California Institute of Technology. Most of the POOMA developers have recently left.
4. **Maturity**—The POOMA project started around 1994. (POOMA 2.x is currently under development.) POOMA has achieved one of its requirements (i.e., code portability across serial, distributed, and parallel architectures with no change to source code).
5. **Users**—Most users are within the LANL.
6. **Language**—The tools in POOMA are implemented as a C++ class library. Application development in C++ using POOMA can be done efficiently by using and/or deriving from those classes with the C++ features of inheritance and polymorphism. However, legacy codes in Fortran cannot be supported easily in POOMA without major code changes.
7. **Tools/utilities included**—POOMA tools and utilities support data types such as field (i.e., array) and particle. With those basic data types, application codes can use the tools and utilities such as meshes and differential operators.
    - **Fields**—Fields are multidimensional arrays representing grids with definable centering. They may be fully contained within a processing node, or spread across nodes, according to user directives. POOMA predefines field types for the most common element types: scalars (double, integer, etc.), vectors, tensors, and symmetric tensors. POOMA users may specify other arbitrary element types, although this is not completely straightforward.
    - **Particles**—An instance of the particle class actually stands for a set of particles, with user-definable characteristics. Particle sets may also be distributed across nodes, and operations on them are expressed in a data-parallel style.
8. **Application interface**—Based on utilities provided by POOMA, application codes in C++ can be developed quickly through the features of C++ inheritance and polymorphism.
9. **Documentation**—Most information on POOMA can be found at [1].

10. **Associated software**—Currently POOMA uses the software package "Cheetah" to wrap Message Passing Interface (MPI) and/or Shared Memory (SHMEM) for message-passing operations and uses the software package "SMART" to perform multithreaded operations. Multithreaded POOMA 2.x programs can be profiled using the Tuning and Analysis Utilities (TAU) library. TAU is a set of tools that allows developers to analyze the performance of distributed and multithreaded programs. It is especially useful in tracking C++ classes that are created and destroyed dynamically. "Cheetah" and "SMART" were developed at LANL.

11. **Special features**—True portability across serial, parallel, and distributed architectures.

2.2.3. Description

POOMA is used to write high-performance Partial Differential Equations (PDE) solvers using finite-difference methods on structured, unstructured, or adaptive grids, particle methods, or hybrid methods which combine the above. An earlier version of POOMA, generally referred to as POOMA R1, has enjoyed considerable success meeting these goals in real applications, which include gyrokinetic particle-in-cell plasma simulation, multimaterial compressible hydrodynamics, accelerator modeling, and Monte Carlo transport. POOMA 2.x is the next generation of the POOMA software, designed to take advantage of advances in C++ compiler technology, multithreaded operation, and a new, highly extensible design.

As indicated at the POOMA Web site, "POOMA has a flexible array class that supports a plug-in 'Engine' architecture to achieve representation independence. It includes a powerful system for specifying and combining domains to construct views of arrays. These views *are* arrays, so they can be used anywhere an array is expected. Using a novel ExpressionEngine abstraction, array expressions in POOMA are also first-class arrays. POOMA supports multithreaded execution on shared-memory multiprocessors using the SMARTS runtime system. An experimental asynchronous scheduler is available that uses data-flow analysis to perform out-of-order execution in order to improve cache coherency. POOMA hides the details of parallel computation in a flexible Evaluator architecture. For the user, this means that a program can be written in a highly abstract data-parallel form, tested and debugged in serial mode, and then run in parallel with very little effort. With POOMA version 2.1, physics abstractions like fields, coordinate systems, meshes, efficient differential operators, and particles have been introduced." [1]

POOMA programs are written at a high level, using data-parallel array expression in the style of High Performance Fortran (HPF) or at a serial level using iterators on each CPU. They can achieve high performance (comparable to Fortran) thanks to a clever compilation technique called expression templates [4]. Moreover, POOMA programs achieve true portability across serial, parallel, and distributed architectures.

One of the principal motivations behind POOMA is to provide C++ classes that directly address numerical science problems using the methodology of numerical scientists. By managing boundary conditions, and supporting efficient evaluation of differential operators, these classes provide the functionality that modern numerical algorithms require, and allow numerical scientists to concentrate on what they want to calculate, rather than on how it is to be calculated.

The main goals of the POOMA framework include the following:

- Code portability across serial, distributed, and parallel architectures with no change to source code;
- Development of reusable, cross-problem-domain components to enable rapid application development;
- Code efficiency for kernels and components relevant to scientific simulation;

- Framework design and development driven by applications from a diverse set of scientific problem domains;
- Shorter time from problem inception to a functional parallel simulation.

POOMA is an object-oriented design framework with a layered structure. Figure 1 below shows the basic structure of the POOMA framework. Application programmers can use and/or derive from these C++ classes, which present a data-parallel programming interface at the highest abstraction layer. Lower implementation layers encapsulate distribution and communication of data among processors.

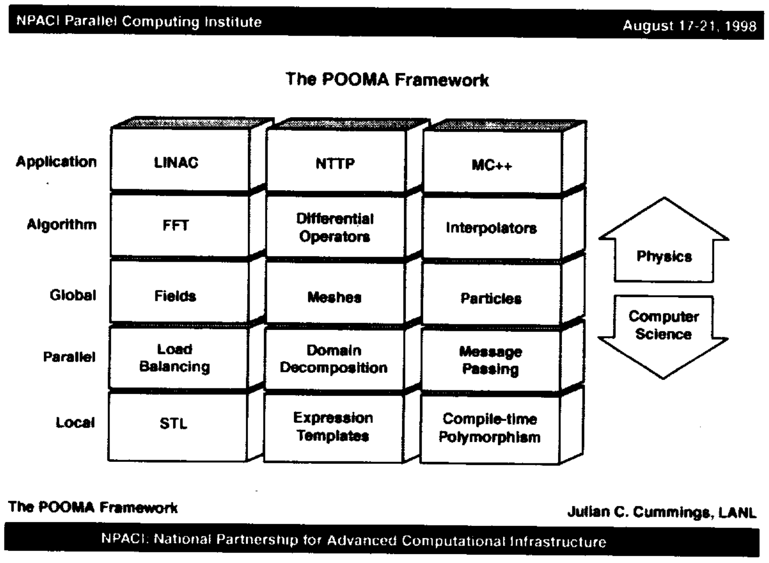

Figure 1. The POOMA Framework Design [1]

As mentioned above, POOMA has used expression templates to optimize performance. By using expression templates, it has attempted to solve the problem of poor performance that is inherent in the C++ language. The following example shows one of problem origins:

```
class Matrix { /*...*/ };
Matrix A, B, C, D;
/* ... */
A = B + C + D;
```

Suppose that class matrix overloads the operators "+" and "=" to perform element-wise addition and assignment. The final line will be evaluated in a series of binary operations. These will involve temporary matrix objects that store intermediate results: tmp1 = B + C; tmp2 = tmp1 + D; A = tmp2. Creating and destroying temporary objects can severely degrade performance, especially if each object contains a significant amount of data. This problem has been recognized for some time, and various attempts have been made to solve it. The best solution to date is expression templates [4], a flexible and general device that avoids the creation of temporary objects. POOMA relies heavily on expression templates to optimize data-parallel expressions involving particles and fields. POOMA applications thereby retain the benefits of overloaded operators with no loss in performance. Since the ROSE preprocessor used in Overture is not ready for public release, it is difficult to compare the technique of expression template with the ROSE preprocessor at this time.

Thus far, there are a limited number of scientific demonstrations of POOMA inside the LANL and in some universities. Those applications are multimaterial hydrodynamics, particle-in-cell plasma simulation, Monte Carlo neutronics, accelerator physics, volume fraction and programmed burn, and numerical tokamak. All these applications have been developed in close collaboration with the POOMA development team.

2.2.4 Evaluation

Strengths

POOMA application programs are written at a high level, using data-parallel array expressions in HPF style or at a serial level using iterators on each CPU. Those programs can achieve high performance (comparable to Fortran) by using a compilation technique called expression templates. Moreover, they achieve true portability across serial, parallel and distributed architectures. The syntax is similar to that of Fortran 90: a single assignment fills an entire array with a scalar value, subscripts express ranges as well as single points, etc. Since the expression template is a feature of C++, the technique of expression templates can be supported by a standard C++ compiler.

Weaknesses

The learning curve for POOMA may be steep because of its extensive use of C++ template features. POOMA intends to be used for writing a new code in C++ not for adopting a legacy code written in Fortran. The biggest problem with POOMA is that the compiling time can be extraordinarily long (hours). Most compilers produce long and cryptic error messages if they encounter an error while expanding template functions and classes, particularly if those functions and classes are nested. As POOMA uses templates extensively, it is not uncommon for a single error to result in several pages of error messages from a compiler. Finally, some debuggers still provide only limited support for inspecting template functions and classes. All of these problems are actively being addressed by vendors, primarily in response to the growing popularity of the Standard Template Library (STL).

Summary

The POOMA framework has successfully achieved portability across serial, parallel, and distributed architectures. However, the extraordinarily long compiling time due to the use of expression templates technique makes POOMA difficult to use for practical applications. But its object-oriented design and the methodology used in achieving portability across platforms can be useful for developing an ESMF.

2.2.5. References

1 URL: http://www.acl.lanl.gov/pooma/

2 URL: http://www.physics.ucla.edu/icnsp/Html/mark.htm

3 URL: http://acts.nersc.gov/pooma/main.html

4 Veldhuizen, Todd, 1995. "Expression Templates." C++ Report 7:5 (June, 1995), 26–31. Reprinted in C++ Gems, Stanley B. Lippman, ed., 1996. Sigs Books, NY.

2.3 Cactus

2.3.1 Introduction

Cactus [1] is a software framework resulting from the collaboration of numerous institutions including the Albert Einstein Institute (AEI) [2], Washington University Gravity (WUGRAV) group [3], National Center for Supercomputing Applications (NCSA) [4], Konrad-Zuse-Zentrum fuer Informationstechnik Berlin [5], International Numerical Relativity Group (INRG) [6], Rechenzentrum Garching (RZG) der Max-Planck-Gesellschaft [7], Argonne National Laboratory (ANL) [8], and Universitat de les Illes Balears (UIB) [9].

As described on the Cactus Web page,

> "Cactus is an open source problem-solving environment designed for scientists and engineers. Its modular structure easily enables parallel computation across different architectures and collaborative code development between different groups. Cactus originated in the academic research community, where it was developed and used over many years by a large international collaboration of physicists and computational scientists.
>
> "The name Cactus comes from the design of a central core (or flesh) which connects to application modules (or thorns) through an extensible interface. Thorns can implement custom-developed scientific or engineering applications, such as computational fluid dynamics. Other thorns from a standard computational toolkit provide a range of computational capabilities, such as parallel input/output (I/O), data distribution, or checkpointing.
>
> "Cactus runs on many architectures. Applications, developed on standard workstations or laptops, can be seamlessly run on clusters or supercomputers. Cactus provides easy access to many cutting-edge software technologies being developed in the academic research community, including the Globus Metacomputing Toolkit, HDF5 parallel file I/O, the PETSc scientific library, adaptive mesh refinement, Web interfaces, and advanced visualization tools." [1]

2.3.2 Synopsis

Prominent information about Cactus includes the following:

Community of origin—The Cactus framework originates in the relativistic physics community.

1. **Description**—Cactus is a component-oriented application development environment for scientific computing which employs a common interface to components written in multiple languages.

2. **Team**—The Cactus team includes at least 25 individuals, a partial list of which includes Gabrielle Allen [10], Tom Goodale [11], Ed Seidel [12], Thomas Radke [13], Gerd Lanfermann [14], and others [15]. The team is actively functioning and is anxious to work with users from other communities to further enhance the code infrastructure.

3. **Maturity**—As of August 2000, the Cactus team is shipping version 4.0, beta 8.0. Version 1.0 was released on April 24, 1997. Cactus spans at least 3 years and four major releases. The predecessors of Cactus go back to the early 1990's. Cactus benefitted from the NSF Black Hole Grand Challenge project and several other attempts to develop a framework for research groups. Cactus has a long heritage in community framework development.

4. **Users**—Cactus has approximately 100 users, mostly within the relativistic physics community, but recently expanding into other disciplines, such as aerodynamics, condensed matter, chemical engineering, geophysics, and climate modeling. Some of the users, as noted on the Cactus home page, are listed below:

   - Within the relativistic physics community, application groups actively using Cactus include the following:
     - The AEI [2] has developed a Cactus Relativity Toolkit.
     - The WUGRAV [3] has developed several Cactus modules.
     - University of Pittsburgh
     - The Pennsylvania State University numerical relativity group is using Cactus for its Agave [16] code in performing 3D evolutions of spacetime.
     - University of Texas
     - University of Timisoara (Romania)
     - University of Wien
     - University of Washington, Seattle
     - Several groups in Italy
     - Physical Research Lab and Raman Research Institute (India)
     - University of California, Santa Barbara
     - University of Southampton
     - University of Portsmouth
   - Astrophysics users include the following:
     - The cosmology group at NCSA [4], including Michael Norman [17], are porting the successful Zeus code to Cactus. Zeus has several hundred users worldwide.
     - The European Union Network Project [18], a project simulating collisions of neutron stars and black holes, which is closely related to the NASA Neutron Star Grand Challenge project [19].
     - Max-Plank-Institut for Astrophysics
     - The Astrophysics Simulation Collaboratory (ASC) [20] is a project sponsored by National Science Foundation (NSF) grant PHY 99-79985 to Rutgers University, University of Chicago, University of Illinois, and Washington University, that enables large-scale simulations in astrophysics. The code is based on Cactus. A progress report [21] provides insight into how Cactus can be modified to support new simulation applications.
   - Aerospace users include the following:
     - Deutsche Luft und Raumfahrtzentrum [22], the German Aerospace Center, is working with an industrial partner to plug an aerospace application into Cactus.
     - Condensed matter physics users include James Sethna [23] and his condensed matter physics group at Cornell University.
     - Chemical engineering groups include Ken Bishop [24] and his chemical engineering group at the University of Kansas.

  - Geophysics groups using Cactus include Bosl [25] and the geophysics group at Stanford University are planning to use Cactus. Bosl is also associated with the Digital Earth project [26], a project to develop object-oriented coupled models for geophysical modeling.

- Computational science groups that are actively using and/or developing Cactus include
  - NERSC [27] at Lawrence Berkley Laboratories (LBL)
  - NCSA [4]
  - The University of Chicago
  - Clemson University
  - GMD-First in Berlin is working to incorporate Janus, an unstructured mesh code, as a driver layer.
  - Mike Holst's [28] group at University of California, San Diego (UCSD)
  - Charlie Crabb's [29] group at Lawrence Livermore National Laboratory (LLNL) [30]
  - Imperial College, London, plans to develop an expert system to aid in connecting thorns to build an appropriate application.
  - Most recently, Cactus was ported [31] to the NT Cluster at the Cornell Theory Center (CTC) [32].

- Cactus, having a long history of grid computing experiments, is becoming more extensively used in grid communities such as the following:
  - European Grid Project (EGRID) [33]
  - U.S. Grid Forum [34]
  - The Grid Application Development Software Project (GrADS) [35], a project sponsored by the NSF Next Generation Software programs to simplify distributed heterogeneous computing, directed by Ken Kennedy [36], uses Cactus as a test grid application.
  - Globus [37] developers
  - The German Gigabit Testbed Project (TIKSL) [38] uses Cactus for their simulation environment.

1. **Language**—The core of Cactus is written in ANSI C, with PERL heavily used during the configuration process. There is no Fortran or C++ in the core code. Cactus currently accepts thorn modules in Fortran 77, Fortran 90, C, and C++. Support for other languages such as Java, Practical Extraction and Report Language (PERL), and Python is planned in the next version release (4.1).

2. **Tools/utilities included**—Cactus comes with several toolkits, mostly oriented toward relativistic physics. It also has a runtime system with a scheduler.

3. **Application interface**—Cactus employs a custom common interface to thorns (component modules) which is language independent and provides some object-oriented capabilities.

4. **Documentation**—The Cactus team actively maintains a Web site [1], which provides links to a host of information. Printed documentation includes a user's guide [39], quick start guide, and a set of developer tutorials [40], [41]. Additionally, the Web site provides access to numerous presentations [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], publications about Cactus [55], [56], [57], [58], [59], and application publications and posters [60], [61], [62], [63], [64], [65],

[66], [67], [68]. Cactus has also been the focus of a workshop [69], [70] held at NCSA [4]. Thus, Cactus has a significant amount of associated documentation in its user community.

5. **Associated software**—Cactus requires PERL [71], Gnu Make [72] and a C/C++ compiler. It also interfaces with the following software:
   - Concurrent Version Systems (CVS) [73]—Source code configuration management
   - Cygwin [74]—Unix utilities for Windows platforms
   - MPI and MPICH [75]—Message-passing interfaces for parallelism
   - Globus [76]—Toolkits for computational grids
   - FlexIO [77]—Application program interface (API) for storing multidimensional scientific data
   - HDF5 [78]

   —File format for storing scientific data
   - IEEEIO [79]—library for storing multidimensional data in binary format
   - LCAVsion [80]—scientific visualization tool
   - Amira [81]

   —Scientific visualization tool
   - Portable Extensible Toolkit for Scientific Computing (PETSC) [82]—suite of data structures and routines for scalable (parallel) solution of PDE problems.
   - Grid Adaptive Computational Engine (GrACE) [83]
   - Autopilot [84]

   —Real-time adaptive resource control
   - OpenDX [85]—visualization
   - Performance Data Standard and API (PAPI) [86]—performance monitoring library
   - Panda [87]

   —Parallel I/O library

1. **Performance**—So long as most of the execution time is spent inside a module/thorn, performance of a climate application using Cactus should be about the same as before. The Cactus scheduler does not play a large role in the execution time. Cactus achieved 142 Gflops on a 1024 Cray T3E-1200 with a full thorn set solving the coupled Einstein-Hydro equations.

2. **Special Features:**
   - Unique component interface spans Fortran and C and supplies inheritance capability for data declared using the interface.
   - Cactus applications can include component modules that allow them to function as Web servers [88]. The running application can be inspected via a Web browser through which the user can see the configuration of the application, the currently running modules, the values of the parameters, and graphics showing intermediate results. Application parameters can be configured as steerable, enabling them to be changed via the Web browser during the application run. The Cactus homepage [1] provides a link to a perpetual Cactus run.

- Cactus is especially strong in remote visualization, steering, and grid computing and is used heavily in development of these new technologies worldwide.
  - The Web server now allows fine control of the simulation, such as pausing, terminating, and checkpointing.
  - The Web server interface is being currently developed and will increase in features and functionality.
  - Applications developed with Cactus are automatically grid-enabled and can make use of advances in grid computing.
  - There are several projects and modules for remote access to visualization and control.
  - An advanced portal for Cactus is being developed at ANL, in collaboration with researchers at LBL, NCSA, Washington University, and AEI, to facilitate all aspects of Cactus configuration, parameter input, job management, remote submission on resources, and job monitoring and steering. This capability will be demonstrated at the Supercomputing 2000 Conference.
  - With respect to rewriting codes to use Cactus, there are several levels of integration. On the low end, there is little work and virtually no code intrusion. An entire code may be plugged in but may only use parallelism. At the high end, with more work, an existing code makes intense use of the Cactus architecture. In this respect, the rewrite effort scales the same way. The code can be shifted from the low end to the high end, as the programmer wishes.

2.3.3 Description

High-Level Design

Because of the high complexity of solving relativistic equations [58], described as "among the most complicated seen in mathematical physics," members of the relativistic physics community created Cactus as a common development tool which allowed multiple components, known as thorns, to be assembled and run in the same environment. The need for a modular system is driven by the desire to have members of the community work relatively independently on various software components. The need for multiple language support is derived from the fact that community members are developing components in various languages, including Fortran 77, Fortran 90, and C. Thus, these needs, in conjunction with other needs, translate into a need for a single development environment with a common interface that can span multiple language modules. The Cactus high-level design is governed by the following major ideas:

1. **"A framework is a reusable semi-complete application that can be specialized to produce custom applications."** [89]

   Cactus is not just a toolkit, a library, or a module that performs a particular function. It is a complete framework that meets the definition criteria, oriented towards the production of complete applications consisting of interacting objects.

2. **Fortran is a crucial scientific language that must be supported because speed is important.**

   John Backus, a Fortran designer, was quoted in *The Grid* [90], [91] (p. 184) as saying the following with respect to the Fortran language:

   > "It was our belief that if Fortran, during its first months, were to translate any reasonable 'scientific' source program into an object program only half as fast as its hand-coded counterpart, then acceptance of our system would be in serious danger... To this day I believe that our emphasis on object program efficiency rather than on language design was

basically correct. I believe that had we failed to produce efficient programs, the widespread use of languages like Fortran would have been seriously delayed." [92]

For similar reasons, rather than abandoning Fortran and choosing an object-oriented language that is much more suitable to the design of their system, the Cactus team has put a great deal of effort into accommodating existing Fortran code.

3. **Modules should be self-contained with small interfaces where all communication takes place through the argument list.**

   James Hack [93], in a discussion on modularization and coupling of climate models said the following:

   > "Historically, scientific progress in atmospheric modeling has been slowed by technical difficulties of incorporating and testing physical parameterizations in different large-scale numerical models. Such problems are fundamentally linked to the fact that most codes are not modular in their design and often make use of very different data structures. Recognizing this problem, a number of scientists from major atmospheric modeling institutions have adopted a set of coding rules to make physics packages more plug-compatible in order to facilitate their easy exchange [94]. Although this set of rules is specifically intended for physics packages that do not have a large number of their own prognostic variables, the conceptual approach is appropriate, and will ultimately prove necessary, for coupling complex climate system components (ocean circulation models, chemical models, biosphere models, etc.). Some of the more important concepts contained in this coding standard are
   >
   > - Each component should refer only to its own subprograms and a limited set of standard intrinsic functions.
   > - Each component should provide for its own initialization of static information and initial data through a single initialization entry point.
   > - All communication between packages shall take place though the argument list associated with a single unique entry point into each package.
   >
   > This approach to coupling major model components is quite reasonable." ([95], p. 317).

   Cactus works best with modules written in this style, where all communication with a module takes place through a single entry point using only parameters in the argument list. Modules using other means of communication or with multiple entry points will have more difficulty working with the Cactus system.

Cactus effectively integrates these three ideas at a high level.

Use of the System

The enabling technology in Cactus is associated with the need for language-independent parameter and data passing. In Cactus, this is accomplished by having the user

1. Create special files in the module directory, independent from the source code, to declare and specify dominant parameters and data. These files provided a means to create data "objects" using inheritance and with public, private, and protected variables. Thorns may be scheduled in groups, with other routines/groups being scheduled before or after. A "schedule while (condition)" construct is also allowed to allow for some dynamic control and for loops. This is more primitive than a

scripting language, except for dynamic change at runtime, but a scripting language is planned for the 4.1 release.

2. Insert special Cactus keywords in the source code subroutine calling parameter lists instead of variable names.
3. Compile the application using the Cactus system.

Cactus, in turn, builds the application by

1. scanning the special files to learn about the variables;
2. inserting variable names automatically in the calling lists for each subroutine;
3. compiling the final application using standard compilers.

When the application runs, Cactus performs the following functions:

1. loads thorns/modules;
2. schedules functions from thorns (note that there is more than one entry point to a thorn. They are not treated as monoliths.);
3. transfers data between thorns;
4. provides interfaces to visualization tools;
5. provides interfaces to external Web browsers;
6. writes data.

The enabling technology, then, is the Cactus source code, PERL [71] scripts, and interface conventions which enable it to accomplish these tasks. PERL is a language that is traditionally used for text processing because of its powerful pattern recognition capabilities. With Cactus, PERL and the C-preprocessor (CPP) are used as part of the build process but are not part of the runtime. Cactus provides easy parallelization for modules.

Views of the System

Cactus is a comprehensive system that can be described from several perspectives, each providing insight into what Cactus can do:

- As a language extension to C and Fortran.
- As an object-oriented language.
- As a compiler.
- As a development environment.
- As a runtime environment.
- As a language extension.

As Cactus requires the user to insert special keywords in the source code, Cactus is a type of **language extension**. From this perspective, Cactus is similar in approach to High Performance Fortran ([96] and [91], p. 188), an extension to Fortran in which areas of data parallelism can be denoted in the code via comments or several new keywords. Cactus is different from HPF in that the new keywords do not denote regions of parallelism but instead denote regions of the code where Cactus needs to substitute variable names at compile time. In a similar vein, whereas the principal focus of NPF is the layout of arrays ([91], p. 188), one of the main focuses of Cactus is storing array information in objects through an inheritance mechanism. However, Cactus is not an extension to a single language (such as Fortran), but is instead an extension to *multiple* languages (Fortran and C) using the same keywords. Because multiple languages are involved, Cactus also specifies unique names for types such as CCTK_REAL which corresponds to a real in Fortran or a double in C.

An Object-Oriented Language

As it provides the means for users to build data structures in terms of other data structures using an inheritance mechanism, Cactus is a type of **object-oriented language**. Inheritance is a powerful feature of object-oriented languages and a key part of object-oriented frameworks ([89], p. 5). Inheritance is missing from both Fortran and C. Cactus currently provides this inheritance for data structures but not for method/ function names, thus allowing "subclasses" to be derived from previously defined data structures. Inheritance for function names will be available in the final release via function aliasing.

In addition to having inheritance, Cactus is like object-oriented languages in that thorns are similar to classes both in the manner of specification and in operation. For example, for a hypothetical *MyClass* class definition in the C++ language, the class variables are traditionally specified in two files: (1) MyClass.h, containing declarations, and (2) MyClass.C, containing member functions. Each member function of MyClass automatically has access to all data elements declared as part of the class in MyClass.h without passing in the function argument list. If the programmer decides that MyClass requires an additional data member, such as a gridded three-dimensional array, Z, then Z is added to MyClass.h and becomes automatically available to all member functions of MyClass. Thus the language provides the means to "pass" class variables to member functions without changing the function prototypes. In a similar fashion Cactus provides a means to declare "member data" in files separate from the thorn source code and automatically "pass" these to the thorn functions without requiring the user to change the function prototype.

Though Cactus is similar to object-oriented languages in form and function it is unlike them in elegance. Cactus is awkward in both form and function. Instead of a single declaration file (MyClass.h), three are required (interface.ccl, param.ccl, and schedule.ccl). Instead of meaningful type names like real or double, awkward names such as CCTK_REAL are used for Cactus-owned variables. Some of this awkwardness is unavoidable, a direct byproduct of trying to make something that interoperates between two dissimilar products. In the same way that a stereo system designed to operate in either a bus or a motorcycle, with two different mounting environments and two different acoustical environments would be awkward, the Cactus design is awkward because it straddles two different languages, thus requiring awkward conventions. Nevertheless, even the awkwardness can be a benefit in some cases, as CCTK_REAL can be defined as double or float, enabling users to easily configure the degree of precision in a computation.

A Compiler

In the sense that Cactus scans source code and looks for keywords, it is a type of **compiler**. Cactus uses GNU make and PERL scripts to scan the source code, perform substitutions and arrange for the overall build of the application. It also relies on the CPP to perform macro substitutions. Because it calls the other compilers, the CPP is a type of super-compiler that spans the multiple languages.

A Development Environment

As it manages the production of the final application, Cactus is a type of development environment. The user specifies code in thorns and Cactus provides assistance in assembling the components.

A Runtime Environment

As it provides scheduling of the various modules during program execution, Cactus is a type of **runtime environment**. Cactus does not just compile source code written by somebody else, it also inserts specific code of its own to provide runtime scheduling between the different thorns. At a high level this is similar to the Parallel Application Workspace (PAWS) environment [97] where a central controller coordinates the running of different components. At a low level this is similar to a language like Java which generally runs on a virtual machine which creates, schedules, and controls the objects. This does not mean that there are two levels of control. It means that Cactus can be viewed in these two ways.

2.3.4 Evaluation

Cactus is unique among the frameworks we examined because of its comprehensive modeling approach. It is worthy of close consideration by the climate community, not only for its implementation, *but more importantly for its method.* In many respects, the needs of the climate community are not significantly different than the needs of the relativistic physics community or of any other community that develops and uses complex software and computer systems. The comprehensive manner in which Cactus addresses these universal needs provides valuable insights into what a framework should be and can become.

Cactus addresses the issue of how to promote the development of interchangeable software within a community by providing a tool with a comprehensive focus. The genius of the Cactus framework is not so much in the individual software elements as in the comprehensive nature of the solution, which enables the business of assembling and running an application to be effectively conducted. Individually, the elements may be either highly desirable to the climate community or, alternately, distasteful and awkward. But even some of the awkward elements, as part of a larger system, have a strength that other frameworks do not provide.

Strengths

These are the issues that the Cactus framework addresses:

1. Cactus is a framework in the true sense. It addresses the need for a community software focal point by defining the nature of the software components and the relationships ([89] p.4) between them and by providing a common tool to assist with assembly and scheduling.

2. According to the pattern of true frameworks, Cactus provides a mechanism for *inversion of control* ([89] p.5), a supervisory process (flesh) which schedules the interacting objects (thorns), thus relieving the module developer of that responsibility.

3. Cactus addresses the reality of researchers working independently by developing the thorn/arrangement concept ([39], Chapter B1) which spans code, interface, and documentation. These concepts provide a common format to use that makes code more interchangeable and understandable among members of the community.

4. Cactus addresses the reality of researchers working on a range of platforms because it is relatively platform independent, running on platforms ranging from laptops to supercomputers.

5. Cactus simultaneously addresses two of the most difficult realities facing scientific computing today:

    - There is a large body of working, debugged, efficient legacy Fortran code with which researchers are intimately familiar.

- The majority of training and new developments in computer science are taking place in newer languages such as C, or object-oriented languages such as C++.

Cactus addresses these issues by defining a single interface that can be used with either language. The interface includes definitions of data type and array structures, and provides rudimentary inheritance capabilities that can be used with Fortran 77, Fortran 90, or C code. Thus, Cactus provides access to some object-oriented features without forcing users to migrate all of their code to an object-oriented language. This is a much better solution than defining an interface at the lowest level, corresponding to the intersection of languages.

6. Cactus addresses the need for researchers to inspect a running process via the Web [88]. Cactus provides utilities that allow Web browsers to connect to a running process for inspection. The Cactus home page [1] provides a sample connection. It is important to note that it is not only possible to inspect via a browser, but that it is also possible to retrieve any data from the memory of a running simulation and have it streamed in HDF5 format to a local host, for more in-depth visualization and analysis.

7. Cactus has a strong role in the emerging area of Grid computing because it enables applications, which are not Grid aware, to run in a Grid environment, thus saving the researchers the effort of building in this ability.

Weaknesses

Cactus has the following weaknesses:

1. The standard Cactus arrangements [98] include little code that is directly applicable to the climate community because they are oriented towards Cartesian grids instead of geographical grids. Current arrangements include
   - Base—boundary conditions, Cartesian coordinates, symmetry conditions, general I/O, scalar and screen output, time step
   - Elliptic—solvers for elliptic equations
   - PUGH—parallel driver layer based on MPI, three-dimensional parallel interpolator
   - PUGHIO
   - CactusWaveToy—Wave Evolver used as a demo
   - Net
   - External

   Therefore, the climate community would have to develop/adapt their own code and modules. Nevertheless, as demonstrated by the spectrum of users, the Cactus team is anxious to work with communities like the climate community to extend Cactus functionality in specific ways.

2. The scheduler may not be as flexible as desired for some climate simulations. Some climate simulations run models sequentially with different numbers of iterations for each model with intermediate coupling exchanges. This can be accommodated to some degree using the WHILE functionality in Cactus. Nevertheless, having a real scripting language would provide more powerful capabilities.

3. The common interface between Fortran/C is awkward. By moving the subroutine variables out of the subroutine into separate files it becomes harder to see what is being passed in and out.

Nevertheless, hidden arguments are only those associated with the scheduler. All other arguments can be declared in the code as before.

4. Saying that Cactus works with Fortran, C, etc., is slightly misleading. Once a code is converted to Cactus style it is committed and can be used on no other platform. If the Cactus-modified interface subroutine is small, then all of the lower-level subroutines can be pure Fortran or C so this isn't a problem. At the same time, however, this limits the benefits of data "objects" because they have to be immediately decomposed into variables and passed around. On the other hand, if the choice is made to pass around data as Cactus "objects" at all levels, then the code is fully Cactus-ized and will not compile or work on any other platform. Thus, when a community chooses Cactus, it must do so with a full commitment. This is a weakness only in cases where interchanging code with non-Cactus communities is important or where Cactus cannot be a complete solution for the climate community. Where such an exchange is necessary, one way to get around this problem is to define a macro to refer to CCTK_ARGUMENTS when using Cactus and to refer to the actual list of parameters when not using Cactus.

5. Cactus does not have a scripting language, as does ROOT [99], a capability that could be helpful. This capability is planned for release 4.1

6. The type of inheritance supplied by Cactus is not the same type of inheritance that is integrated into object-oriented programming languages. Therefore, it has a limited ability to solve the types of problems that inheritance is typically used for.

Summary

Cactus is a complete framework solution to scientific modeling. It addresses the issue of differences between Fortran and C with a common interface and provides a development environment and a runtime environment, including a scheduler, for the modules. It also provides a means to inspect running code. It has a large amount of documentation, is used by many organizations and is stable, and has a supportive development team. The main disadvantage with Cactus is that it may require some adaptation before it can be used by the climate community. Nevertheless, the Cactus team is highly motivated to work with the climate community and other communities to extend Cactus capabilities.

2.3.5 References

[1] Cactus. URL: http://www.cactuscode.org

[2] Albert Einstein Institute (AEI). URL: http://www.aei-potsdam.mpg.de

[3] Washington University Gravity Group (WUGRAV). URL: http://wugrav.wustl.edu/

[4] National Center for Supercomputing Applications (NCSA).
URL: http://www.ncsa.uiuc.edu/newtest/NCSA/whatisncsa.html

[5] Konrad-Zuse-Zentrum fur Informationstechnik Berlin (ZIB). URL: http://www.zib.de

[6] International Numerical Relativity Group (INRG). URL: http://jean-luc.ncsa.uiuc.edu

[7] Rechenzentrum Garching der Max-Planck-Gesellschaft (RZG). URL: http://www.rzg.mpg.de

[8] Argonne National Laboratory (ANL). URL: http://www.anl.gov

[9] Universitat de les Illes Balears (UIB). URL: http://www.uib.es

[10] Gabrielle Allen, allen@aei-potsdam.mpg.de.

[11] Tom Goodale, goodale@aei-potsdam.mpg.de.

[12] Ed Seidel, eseidel@aei-potsdam.mpg.de. URL: http://jean-luc.ncsa.uiuc.edu/People/Ed

[13] Thomas Radke, tradke@aei.mpg.de.

[14] Gerd Lanfermann, lanfer@aei.mpg.de.

[15] Masso, Joan, 1999: Cactus Philosophy, *Cactus Workshop, NCSA*, Sep.
URL: http://www.cactuscode.org/Workshops/NCSA99/talk2/index.htm

[16] AGAVE Code Project (AGAVE). URL: http://www.astro.psu.edu/users/nr/Agave

[17] Michael L. Norman, norman@ncsa.uiuc.edu.
URL: http://www.astro.uiuc.edu/department/faculty/norman.html

[18] EU Network Project. URL: http://www.aei-potsdam.mpg.de/research/astro/eu_network/description.html }

[19] NASA Neutron Star Grand Challenge Project. URL: http://wugrav.wustl.edu/Relativ/nsgc.html

[20] Astrophysics Simulation Collaboratory (ASC). URL: http://wugrav.wustl.edu/ASC/mainFrame.html

[21] Staff, ASC, 2000: NSF Progress Report, Astrophysics Simulation Collaboratory.
URL: http://wugrav.wustl.edu/ASC/report1.html

[22] Deutsche Luft- und Raumfahrtzentrum (DLR). URL: http://www.dlr.de

[23] James Sethna, 607-255-5132, sethna@lassp.cornell.edu.
URL: http://www.lassp.cornell.edu/sethna/sethna.html

[24] Kenneth Bishop, 785-864-2918, kbishop@ukans.edu.
URL: http://www.engr.ukans.edu/cpe-grad/bishop.html

[25] William Bosl, 650-725-5835, bosl@pangea.stanford.edu. URL: http://pangea.stanford.edu/~bosl

[26] Digital Earth Project (Digital Earth).
URL: http://pangea.stanford.edu/~bosl/EarthObjectsPage/EarthObsPage.htm

[27] National Energy Research Scientific Computing Center (NERSC). URL: http://hpcf.nersc.gov/

[28] Michael Holst, 858-534-4899, mholst@math.ucsd.edu. URL: http://www.scicomp.ucsd.edu/~mholst

[29] Charlie Crabb, 925-424-3265, ccrabb@llnl.gov.

[30] Lawrence Livermore National Laboratory (LLNL). URL: http://www.llnl.gov

[31] Benger, Werner, 2000: Porting CCTK to NT at the Cornell Velocity NT Cluster.
URL: http://www.cactuscode.org/Presentations/Cornell2_August00.ppt

[32] Cornell Theory Center (CTC). URL: http://www.tc.cornell.edu/ctc.html

[33] European Grid Project (EGRID). URL: http://www.egrid.org

[34] US Grid Project (GridForum). URL: http://www.gridforum.org

[35] Grid Application Development Software Project (GrADS). URL: http://hipersoft.cs.rice.edu/grads

[36] Ken Kennedy. URL: http://www.cs.rice.edu/~ken

[37] GLOBUS (GLOBUS). URL: http://www.globus.org

[38] German Gigabit Testbed Project (ZIB_TIKSL). URL: http://www.zib.de./Visual/projects/TIKSL/

[39] Staff, Cactus, 2000: Cactus 4.0 Users' Guide.
URL: http://www.cactuscode.org/Documentation/UsersGuide_html/UsersGuide.html

[40] Team, The Cactus, 2000: Cactus 4.0 A First Tutorial: The 3D Scalar Wave Equation.
URL: http://www.cactuscode.org/Tutorial/sld001.htm

[41] Allen, Gabrielle, 1999: A First Tutorial of Cactus 4.0 - The 3D Scalar Wave Equation. URL: http://www.cactuscode.orgs/Tutorial/index.html

[42] Allen, Gabrielle, 2000: The Cactus Code: A Parallel Collaborative Framework for Large Scale Computing. URL: http://www.cactuscode.org/Presentations/SciComp_August00.sdd

[43] Allen, Gabrielle, 2000: The Cactus Code: a Problem Solving Environment for the Grid, *HPDC 9, Pittsburgh*, Aug. URL: http://www.cactuscode.org/Presentations/HPDC9_August00.ppt

[44] Dramlitsch, Thomas, 2000: Exploring Distributed Computing Techniques with Cactus and Globus, *Globus Retreat*, Aug. URL: http://www.cactuscode.org/Presentations/GlobusRetreat_August00.ppt

[45] Seidel, Ed, 2000: Cactus/TIKSL/KDI/Portal Synch Day, *Cactus Discussion Day, NCSA*, July 28. URL: http://www.cactuscode.org/Presentations/NCSA_July00.ppt

[46] Seidel, Ed, 2000: Cactus in GrADs. URL: http://www.cactuscode.org/Presentations/GrADs_July00.ppt

[47] Lanfermann, Gerd, 2000: Cactus in a Nutshell, *Sun HPC Consortium, Manheim*, Jun. URL: http://www.cactuscode.org/Presentations/SUNHPC_June2000.ppt

[48] Goodale, Tom, 2000: Cactus - The Future, *Supercomputing 2000*. URL: http://www.cactuscode.org/Presentations/Mannheim_June2000.sdd

[49] Allen, Gabrielle, 2000: The Cactus Code: A Framework for Parallel Computing, *Conundrum Series, Lawrence Berkeley Laboratory*, May. URL: http://www.cactuscode.org/Presentations/LBL_May00.ppt

[50] Goodale, Tom, 2000: The Cactus Code for Numerical Relativity. URL: http://www.cactuscode.org/Presentations/Garching_March00.sdd

[51] Goodale, Tom, 2000: Cactus Computational Toolkit. URL: http://www.cactuscode.org/Presentations/Southampton_February2000.sdd

[52] Goodale, Tom, 1999: The Cactus Computational Toolkit. URL: http://www.cactuscode.org/Presentations/FB_September99.sdd

[53] Allen, Gabrielle, 1999: Cactus 4.0, *HPDC8, Redondo Beach*, Aug. URL: http://www.cactuscode.org/Presentations/HPDC8_August99.ppt

[54] Allen, Gabrielle, 1999: MetaComputing Within the Cactus Framework, *Globus Retreat, Rodondo Beach*, Aug. URL: http://www.cactuscode.org/Presentations/GlobusRetreat_August99.ppt

[55] Allen, Gabrielle, Thomas Dramlitsch, Ian Foster, Tom Goodale, Nick Karonis, Matei Ripeanu, Ed Seidel and Brian Toonen, 2000: Cactus-G Toolkit: Supporting Efficient Execution in Heterogeneous Distributed Computing Environments. URL: http://www.cactuscode.org/CacPapers/GordonBell_2000.ps.gz

[56] Allen, Gabrielle, Werner Benger, Tom Goodale, Hans-Christian Hege, Gerd Lanfermann, Andre Merzky, Thomas Radke and Edward Seidel, 2000: The Cactus Code: A Problem Solving Environment for the Grid, *Proceedings of HPDC 9, Pittsburg*, Jun. URL: http://www.cactuscode.org/CacPapers/HPDC9_2000.ps.gz

[57] Allen, Gabrielle, Tom Goodale, Gerd Lanfermann, Thomas Radke and Edward Seidel, 2000: The Cactus Code: A Problem Solving Environment for the Grid, *Proceedings of First Egrid Meeting, Poznan*, Mar. URL: http://www.cactuscode.org/CacPapers/Egrid_2000.ps.gz

[58] Allen, Gabrielle, Tom Goodale, Gerd Lanfermann, Thomas Radke, Edward Seidel, Werner Benger, Hans-Christian Hege, Andre Merzky, Johan Masso and John Shalf, 1999: Solving Einstein's Equations on Supercomputers, *IEEE Computer*, 32, 52-59. URL: http://www.computer.org/computer/articles/einstein_1299_1.htm

[59] Allen, Gabrielle, Tom Goodale, Joan Masso and Edward Seidel, 1999: The Cactus Computational Toolkit and Using Distributed Computing to Collide Neutron Stars, *Proceedings of the Eighth IEEE International*

Symposium on High Performance Distributed Computing, Aug, 57-61.
URL: http://www.cactuscode.org/CacPapers/HPDC8_1999.ps.gz

60 Seidel, Edward and Wai-Mo Suen, 1999: Numerical Relativity as a Tool for Computational Astrophysics, *JCAM*. URL: http://xxx.lanl.gov/abs/gr-qc/9904014_ps.ps

61 Alcubierre, Miguel, Gabrielle Allen, Bernd Bruegmann, Gerd Lanfermann, Edward Seidel, Wai-Mo Suen and Malcolm Tobias, 2000: Gravitational Collapse of Gravitational Waves in 3D Numerical Relativity. *Phs. Rev.*, **D61**. URL: http://xxx.lanl.gov/abs/gr-qc/9904013

62 Arbona, A., C. Bona, J. Masso and J. Stela, 1999: Robust Evolution System for Numerical Relativity. *Phs. Rev.*, **D60**. URL: http://xxx.lanl.gov/abs/gr-qc/9902053

63 Alcubierre, M., S. Brandt, B. Bruegmann, C. Gundlach, J. Masso, E. Seidel and P. Walker, 2000: Test-beds and Applications for Apparent Horizon Finders in Numerical Relativity. *Class. Quant. Grav.*, 17, 2159-2190. URL: http://xxx.lanl.gov/abs/gr-qc/9809004/

64 Font, J.A., M. Miller, W. Suen and M. Tobias, 2000: Three Dimensional Numerical General Relativistic Hydrodynamics: I Formulations. Phs. Rev., **D61.** URL: http:xxx.lanl.gov/abs/gr-qc/9811015

65 Bona, Carlos, Joan Masso, Edward Seidel and Paul Walker, 1998: Three Dimensional Numerical Relativity with a Hyperbolic Formulation. URL: http://xxx.lanl.gov/abs/gr-qc/9804052/

66 Walker, Paul, 1997: Three Dimensional Numerical Relativity with a Hyperbolic Formulation.
URL: http://www.cactuscode.org/CacPapers/cactus1.ps.gz

67 Alcubierre, Miguel, Carsten Gundlach and Florian Siebel, 1996: Using Geodesics to Compare Exact and Numerical Spacetimes. URL: http://www.cactuscode.rog/CacPapers/MA_GR15.ps.gz

68 Alcubierre, Miguel, 1997: Evolution of Brill Waves in 3D Progress Report, *GR15 Meeting, Puna/India.*
URL: http://www.cactuscode.org/CacPapers/BRILL_GR15.ps.gz

69 *High Performance Computing and Cactus Computational Toolkit: A Framework for Solving PDEs in Computational Science with Special Advanced Topics in Astrophysics and Relativity*, 1999, NCSA, Sep.
URL: http://www.ncsa.uiuc.edu//SCD/Training/CactusAgenda.html

70 *Cactus Day at NCSA*, 1999: NCSA, Sep.
URL: http://www.cactuscode.org/Workshops/NCSA99/index.html

71 PERL (PERL). URL: http://www.perl.org

72 GNU (GNU). URL: http://www.gnu.org

73 Concurrent Versions System (CVS). URL: http://www.cyclic.com

74 GNU Utilities for Windows (Cygwin). URL: http://sourceware.cygnus.com/cygwin/

75 Message Passing Interface (MPI). URL: http://www-unix.mcs.anl.gov/mpi/

76 GLOBUS (GLOBUS). URL: http://www.globus.org

77 FlexIO (FlexIO). URL: http://zeus.ncsa.uiuc.edu/~jshalf/FlexIO/

78 Hierarchical Data Format Version 5 (HDF5). URL: http://hdf.ncsa.uiuc.edu/whatishdf5.html

79 IEEE IO (IEEEIO). URL: http://zeus.ncsa.uiuc.edu/~jshalf/FlexIO/IEEEIO.html

80 LCA Vision (LCAVision). URL: http://zeus.ncsa.uiuc.edu/~miksa/LCAVision.html

81 Amira (Amira). URL: http://amira.zib.de

82 PETSC. URL: http://www.mcs.anl.gov/petsc/

83 GrACE. URL: http://www.caip.rutgers.edu/~parashar/TASSL/Projects/GrACE/index.html

[84] Autopilot. URL: http://www pablo.cs.uiuc.edu/Project/Autopilot/AutopilotOverview.htm

[85] OpenDx. URL: http://www.opendx.org

[86] Performance Data Standard and API (PAPI). URL: http://icl.cs.utk.edu/projects/papi/index.html

[87] Scalalable Parallel I/O System (Panda). URL: http://cdr.cs.uiuc.edu/panda/

[88] Benger, Werner, 1999: Web Cactus, NCSA *Cactus Workshop*, Sept 27.
URL: http://www.cactuscode.org/Workshops/NCSA99/talk19/index.htm

[89] Fayad, Mohamed E., Douglas C. Schmidt and Ralph E. Johnson, 1999: Building Application Frameworks. *Wiley*.
URL: http://www.amazon.com/exec/obidos/ASIN/0471248754/qid%3D968778251/102-0715276-9734544

[90] Foster, Ian and Carl Kesselman (Ed.), 1999: The Grid: Blueprint for a New Computing *Infrastructure. Morgan Kaufmann Publishers, Inc.*,
URL: http://www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-475-8

[91] Kennedy, Ken, 1999: Compilers, Languages and Libraries, In *The Grid*, Ian Foster and Carl Kesselman (Ed.), Morgan Kaufman Publishers, Inc., 181-204.

[92] Backus, J., 1978: The History of Fortran I, II, and III. *ACM SIGPLAN Notices*, **13**, 165-180.

[93] James Hack, 303-497-1387, jhack@ucar.edu. URL: http://www.cgd.ucar.edu/asr98/cms.html

[94] Kalnay, E., M. Kanamitsu, J. Pfaendtner, J. Sela, M. Suarez, J. Stackpole, J. Tuccillo, L. Umscheid and D. Williamson, 1989: Rules for the Interchange of Physical Parameterizations. *Bull. Am. Met. Soc.*, **70**, 620-622.

[95] Hack, James J., 1995: Climate System Simulation: Basic Numerical and Computational Concepts, *In Climate System Modeling*, Kevin E. Trenberth (Ed.), Cambridge University Press, 283-318.

[96] Koelbel, Charles, 1996: High Performance Fortran in Practice.
URL: http://www.cs.rice.edu/~chk/hpf-tutorial.html

[97] PAWS. URL: http://www.acl.lanl.gov/paws/

[98] Lanfermann, Gerd, 1999: The Standard Cactus Arrangements, *NCSA Cactus Workshop*, Sept 27.

[99] ROOT. URL: http://root.cern.ch.html

2.4 Overture

2.4.1 Introduction

Overture [1], [2] originates in the Lawrence Livermore National Laboratory (LLNL) and the Los Alamos National Laboratory (LANL). The project is now consolidated into the LANL. Overture is an object-oriented software system for solving Partial Differential Equations (PDE) in serial and parallel environments. It consists of a library of C++ classes for writing serial and parallel PDE solvers using overlapping and/or adaptive block-structured grids. Overture programs are written at a very high-level, using data-parallel array expressions in the style of HPF. It can achieve high performance (comparable to Fortran) through a source-to-source (C++ to C++) preprocessor called ROSE. Effectively, ROSE is a counterpart to the expression template technique of the POOMA framework [3]. This survey is based upon Overture documentation, teleconferences with Overture developers, and the assessment of NERSC on Overture [4].

2.4.2 Synopsis

Prominent information about Overture includes the following:

1. **Community of origin**—Overture originates in the LLNL and the LANL. Now the project is consolidated at the LLNL. .

2. **Description**—Overture provides a portable, flexible software development environment for applications that involve the simulation of physical processes in complex moving geometry such as modeling the motion of a submarine.

3. **Team**—Present members of the Overture team include David Brown, Bill Henshaw, and Daniel Quinlan.

4. **Maturity**—This project began in the early 1990s. The parallel version of Overture is still being developed. All serial implementations require OpenGL (a graphics library). Overture is based on a few other software products, among which are the A++/P++ array library and the ROSE preprocessor. Development of these products is not yet complete. The main issues awaiting resolution are

   - The ROSE preprocessor (needed to ensure adequate performance of aggregate array operations) can currently optimize with hints; eventually it will optimize without hints.
   - Overture runs in a parallel environment, but only for single-grid applications because of some missing features in P++.
   - P++ has only been ported to a network of workstations (Ultra Sparcs), although it should be quite easy to port to most common parallel architectures

5. **User**—most users are inside the LLNL and the LANL. During the last several years, Overture has been used to develop flow solvers for high-speed compressible flow problems, incompressible flow problems, low Mach number and non-Newtonian fluid flow. One National Oceanic and Atmospheric Administration (NOAA) organization at Boulder is trying to use Overture for ocean modeling.

6. **Language**—Overture is written mostly in C++ and the A++/P++ array class library. Using the serial/parallel array class library A++/P++, efficient and portable serial or parallel code is generated. An application program has to be written in C++ if the application program wants to use the features of Overture, especially in the area of parallel communication.

Tools/utilities included—the Overture C++ classes provide tools for the rapid development of application codes. The main class categories are listed below:

- A++/P++ arrays describe multidimensional arrays and provide for serial and parallel operations on those arrays. In the parallel environment, these provide for the distribution and interpretation of communication required for the data parallel execution of operations on the arrays.
- Mappings define transformations such as curves, surfaces, areas, and volumes. These are used to represent the geometry of the computational domain.
- Grids define a discrete representation of a mapping or mappings. These include single grids and collections of grids, in particular composite overlapping grids. The Ogen (a overlapping grid generator for Overture) provides tools for the construction of curvilinear grids, and for overlapping those grids to represent complex moving geometries such as submarine.
- Grid functions provide for the representation and centering of solution values such as density, velocity, and pressure, defined at each point on the grid(s).
- Operators provide discrete representations of differential operators and boundary conditions through finite difference or finite volume approximations.
- Visualization tools based on OpenGL are provided to furnish a high-level graphics interface for visualizing geometry and simulation results.
- Adaptive mesh refinement provides automatic refinement of the overlapping grid structure for increased local resolution and efficiency of computational simulations.
- Load-balancing tools are provided for automatic load-balancing of computations on the adaptive overlapping grid structure on Parallel computers.
- Parallel distribution mechanisms are provided through the PADRE library, part of the Department of Energy (DOE) 2000 ACTS toolkit.

Application interface—based on the utilities described above, application codes written in C++ can be developed quickly by use of the features of C++ inheritance and polymorphism. Since the Overture utilities are based on the A++/P++ array class library which is written in C++, an application cannot be written in Fortran easily.

7. **Documentation**—Overture has a good documentation located at http://www.llnl.gov/casc/Overture

8. **Associated software**—Overture parallelism is implemented using the MPI and PVM message passing libraries, PADRE (a library for the description of distributions of data and the generation of communication schedules to address their communication requirements), and a HPC++ standard threads library (TULIP). In addition OpenGL is used for visualization.

9. **Special features**—the developers of Overture create and use the ROSE preprocessor to optimize C++ performance. In addition, Overture has aggregate array operations (similar to those in POOMA) and tightly integrated graphical features based on OpenGL. AMR++, a package that directly supports adaptive mesh refinement methods, is built on top of Overture. In the future, Overture will add support for unstructured grids and an improved ROSE preprocessor.

2.4.3 Description

The main objective of the Overture project is to provide a flexible code development environment for applications using adaptive overlapping grid technology which addresses:

- complex overset grid data structures;
- need to rapidly develop application codes for existing and future application areas;
- need for software that is portable across multiple architectures from workstations to massively parallel platforms, while maintaining at least Fortran 77 performance.

Overture is an object-oriented framework with a layered structure. Figure 2 shows a block diagram that represents the hierarchy of the class libraries that define the Overture framework. Overture includes about six layers. All levels but the application layer are considered part of the Overture distribution (applications are the property of the application developers in general). C++ optimizing preprocessor, adaptive mesh refinement, turbulence models, and front capturing are still in development and not a part of the public distribution of Overture. Documentation for each library is available on the Web separately. The following is a brief summary of the Overture layers:

- **Communication, data distribution, and threads**—This lowest level within the Overture framework represents a foundation of parallel libraries upon which the rest of Overture is built. This layer includes the MPI and PVM message passing libraries, PADRE, a library for the description of distributions of data and the generation of communication schedules to address their communication requirements; and an HPC++ standard threads library (TULIP).
- **Serial/parallel program interface**—This layer represents the principal interface for the development of applications within Overture. As a single layer of the hierarchy, it represents a hierarchy of program interfaces along with abstractions for the representation of complex geometry, mapping objects; discretizations of the geometric surfaces, mapped grid objects, defined upon those mapping objects; and the representation of data on those grids, grid function objects. Separate container objects are used to represent the collections of each of these mapped grid and grid function objects, thus forming "mappedgridcollection" and "gridcollectionfunction" objects. The details of the representation of the grid function objects are partly encapsulated within the A++/P++ array class library, which forms a fundamental level and abstraction within Overture. The A++/P++ Array Class Library provides the principal mechanism by which parallelism is encapsulated (hidden, to some extent). The array class also is central in addressing the performance issues for different architectures; the work on the array class library specific to performance is considerable. The purpose of the ROSE C++ Optimizing Preprocessor is to isolate the mechanisms by which performance is addressed in the optimization of the array class objects and how they are used within an application and within Overture directly. More details on how performance is addressed within the A++/P++ array class can be obtained from the A++/P++ documentation and related papers [1]. File I/O for all Overture objects is provided via HDF format file structures. The highest level of the hierarchy within the serial/parallel program interface layer is represented by the Operator Library; this class library includes both finite volume and finite difference operators (div, grad, curl, laplacian, different orders of derivatives, etc.). The advantages of this level of abstraction is that it permits the development of applications using high-level code remarkably similar to the fundamental mathematical equations themselves. Finally, the serial/parallel program interface layer includes the graphics and visualization required for real-time graphics, post-processing graphics, movies, and more complex visualization as required for physical simulation using numerical techniques. The visualization understands the geometry represented by the Overture objects so that applications obtain a complete visualization solution.
- **Numerics**—This layer of the Overture framework includes elliptic solver, adaptive mesh refinement, and grid generation applications.
- **Computational Fluid Dynamics (CFD)**—This layer of the Overture framework includes flow solvers, some of which are available within the Overture primer manual and are particularly short and simple to express in the high-level mechanisms represented by the serial/parallel program

interface. Separate libraries isolate current work on turbulence models and front capturing (using level-set methods).

- **Combustion chemistry (interface to Chemkin)**—This layer of the Overture framework represents an interface to the Chemkin software available through Sandia National Laboratory.

  Application layer—This top level defines the highest level of interaction between an application and the Overture framework. An application may, and typically would, interface through multiple levels of the hierarchy represented by Overture. Figure 2 shows a couple of different applications (including weather, ocean modeling) being developed by other groups using Overture and interacting with Overture through different levels of the hierarchy. The developers of Overture did not provide any detailed information on how Overture was used in the climate community. It is not clear from figure 2, but each application additionally interacts with some level of the serial/parallel program interface as well. All internal data (e.g., geometry) is similarly available to C and Fortran applications.

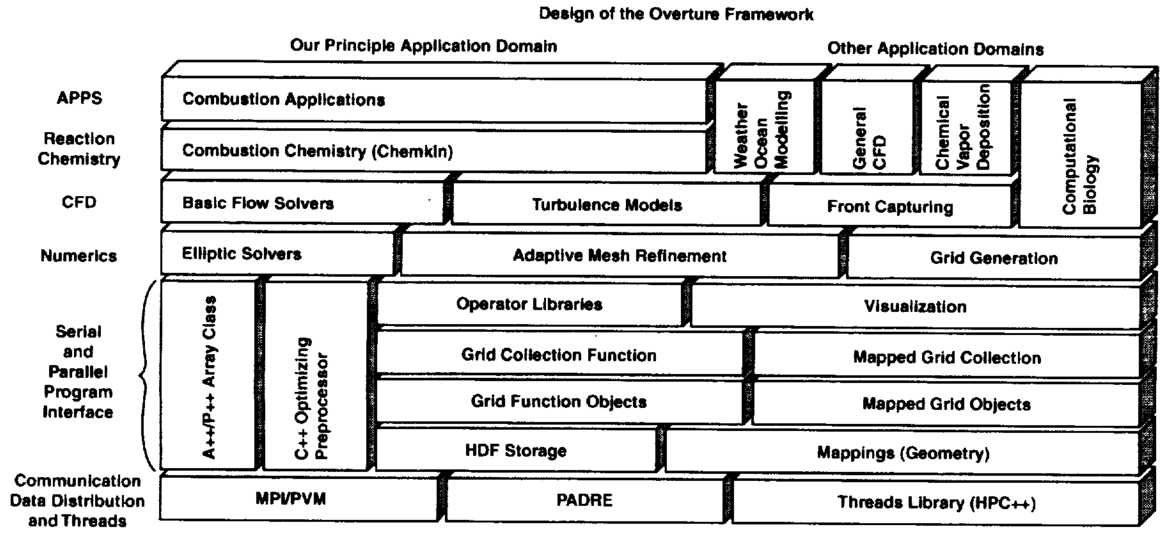

Figure 2. Organization of the Overture Framework [1]

In general, the performance of C++ is not as good as Fortran for numerical computations. To satisfy the requirements of the Overture framework, the performance has been addressed using multiple mechanisms, both internal and external to the C++ language. Three techniques have been investigated:

- **Binary overloaded operators**—Although highly optimized in Overture, the use of binary operators results in only half the performance of optimized Fortran in practice (see figure 3) and performance drops in half again for stencil operations on cache-based machines because reuse of data in cache is poor. However, the method is extremely portable and compiles quickly.
- **Expression templates**—Although the performance is generally superior to binary overloaded operators this is most clear only on stencil operations where the greatest reuse of cached data is possible. But, the expression template technique is only a single statement optimization mechanism and as such it is limited in its applicability. This mechanism uses the C++ template mechanism so aggressively that it lacks portability and results in compile times that increase astronomically (factors of 3,500 times slower) making program development difficult at best.

- **Optimizing preprocessor** (source-to-source transformation)—To avoid the inefficiencies of low-level abstractions within frameworks/libraries, preprocessor mechanisms (see figure 4) have been developed to permit architecture-specific optimizations not possible within libraries. Applications may in the future use mechanisms to permit their optimization using the additional semantics found within the user's use of the framework. The compiler will still not know the semantics of the framework, but the introduction of a third process, an optimizing preprocessor, can provide more aggressive parameterized transformations of the user's expressions using the framework's more restricted semantics. This mechanism uses the Sage II (an object-oriented toolkit for building program transformation systems for Fortran77 including 90, and C, and C++ languages). More information is available at the Sage Web site [5]. On cache-based architecture systems the Overture framework achieves Fortran 77 performance, at a minimum, and can often achieve two to four times better. This work represents the newest addition to the Overture framework and is the result of significant focus on performance as Overture has matured.

Figure 3 shows typical performance for Fortran 77, C++ (overloaded binary operators), C++ with expression templates, and C++ optimized in the manner of the ROSE preprocessor. Part of the research of Overture developers has been in the development and exploration of these different techniques.

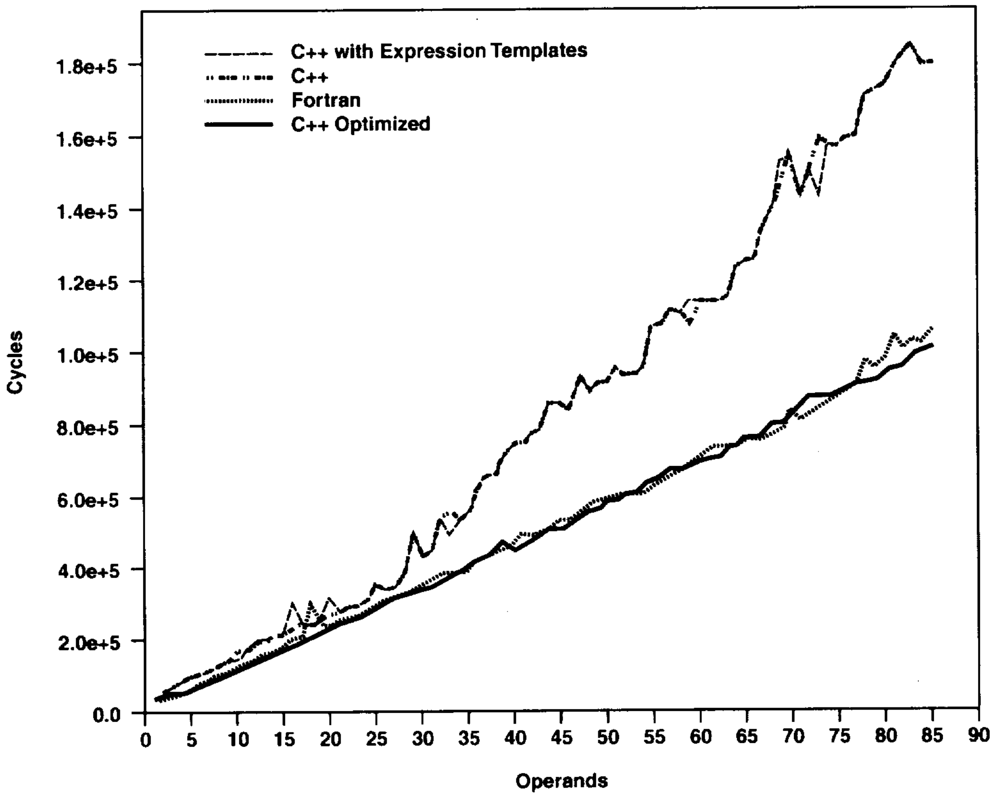

Figure 3. Comparison of Execution Models [1]

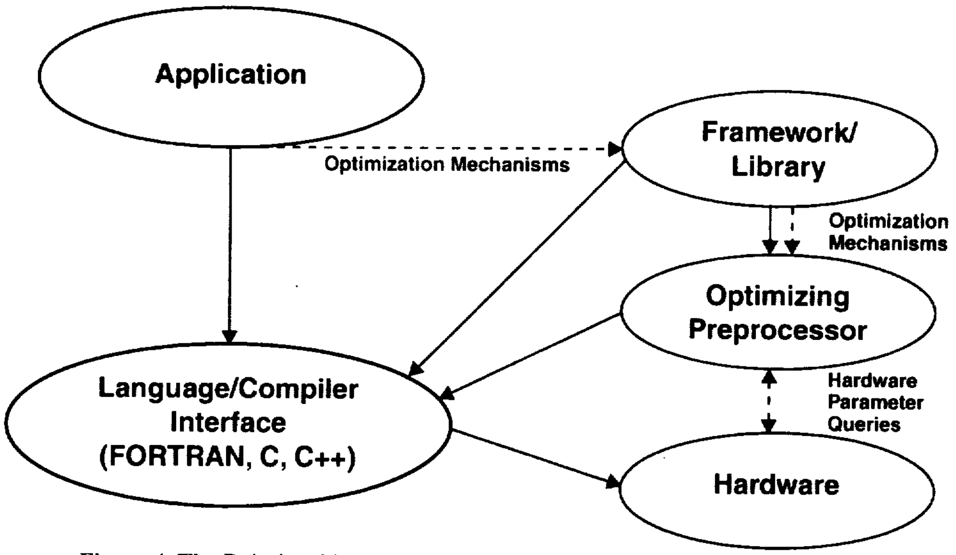

Figure 4. The Relationship of Optimizing Preprocessor with Application, Framework, Programming Languages, and Hardware [1]

The current version of Overture framework has employed the optimizing preprocessor (source-to-source transformation) not the expression template technique.

2.4.4 Evaluation

Strengths

Through its object-oriented design, Overture reduces code duplication, encourages interoperability of application software, and simplifies the learning curve for new computational methods if a user knows C++. Overture's object-oriented architecture provides flexibility to address a wide range of applications that involve simulations of complex moving geometries on serial and parallel computers. The advantages of this approach include reduced code development time and broader, more in-depth research into numerical methods for scientific and industrial applications.

Weakness

Since A++/P++ is the basic data type used in the Overture framework, application codes have to be written with A++/P++ array which means that application codes have to use C++. Legacy code written in Fortran may not be easily ported and supported in the Overture framework. Currently, Overture does not have grid utilities for climate modeling.

Summary

The Overture framework has successfully demonstrated that complex computational problems, such as solving PDE for overlapping grids on parallel computers, can be effectively solved with the help of an object-oriented framework. An application written in C++ can take advantage of the tools and utilities in the Overture framework, while an application written in Fortran cannot do so easily. Currently, the Overture framework does not have the grid utilities for climate modeling. Therefore, a set of geographical grids would have to be developed by deriving from the C++ class library of Overture.

2.4.5 References

[1] URL: http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Bassetti949/

[2] URL: http://www.llnl.gov/casc/Overture/

[3] URL: http://www.acl.lanl.gov/pooma/

[4] URL: http://acts.nersc.gov/overture/main.html

[5] URL: http://www.extreme.indiana.edu/sage/sagexx_ug/sagexx_ug_toc.html

2.5 GEMS

2.5.1 Introduction

Goddard Earth Modeling System (GEMS) is developed by the NASA Seasonal to Inter-annual Prediction Project (NSIPP) and Data Assimilation Office (DAO) organizations at NASA Goddard Space Flight Center. GEMS provides Fortran90 tools/utilities for developing climate modeling applications. A few years ago, DAO and NSIPP worked together to outline the requirements and design for GEMS. Later, DAO and NSIPP separately developed their own versions of GEMS. Since the DAO's version of GEMS is still under development, this report addresses the NSIPP's version of GEMS. This survey is based on information contained at DAO's Web site [1] and review of the GEMS source code of Max Suarez's group.

2.5.2 Synopsis

Prominent information about GEMS includes the following:

1. **Community of origin**—NSIPP
2. **Description**—It provides tools/utilities for climate modeling.
3. **Team**—Max Suarez and others in NSIPP.
4. **Maturity**—It has been used for developing NSIPP's production code.
5. **Users**—NSIPP
6. **Language**—GEMS is written mostly in Fortran 90 and contains some Perl. Applications can be written in Fortran 90.
7. **Tools/utilities included**—The GEMS framework consists of parallel utilities (parallel communication and wrappers), grid utilities (grid manipulation), couplers (exchange information among application components), clock (time and alarm), array (expand and trim array), and other utilities such as conversion of double precision for different computers. In the parallel communication utilities, both SHMEM and MPI message passing protocols are wrapped so that the codes can be run on various multi- or single-processor computer platforms. (Wrapper here means that a new set of instructions is introduced to cover the SHMEM and MPI instructions so that a user does not use SHMEM or MPI directly.)
8. **Application interface**—A group of couplers has been developed to couple various application components (Ocean, Dynamics, Fast Physics, Slow Physics).
9. **Documentation**—Very limited documentation is available (DAO's office Notes [1] and GEMS's source codes).
10. **Associated software**—MPI and SHMEM utilities have been used in parallel communication.
11. **Special features**—GEMS provides basic tools and utilities required for climate modeling.

2.5.3 Description

GEMS developer's interpretation of object-oriented design in the case of global Earth science modeling is as follows:

> "The main goal of object-oriented design is to modularize data, and data transformations, thereby making the system robust to generalizations in the data structures. The object-oriented paradigm seems particularly useful for global Earth science modeling, with models and observations having their own data structures and complex transformations. In a distributed memory environment, a significant portion of the parallel algorithm is associated with transformations from one model grid to another, or from model grid to observation locations. By encapsulating model data structures and model grid transformations, a great deal of code reusability can be realized, at the same time ensuring that changes in one model do not unnecessarily propagate throughout the system." [1]

Based upon this understanding of object-oriented frameworks, the concept of GEMS framework has been developed: "As put forward by Max Suarez, GEMS is made up of a collection of models, each operating on its own grid and on its own state, and a collection of coupler routines which convert from one model grid to another. The resolution and orientation of each model grid is dictated by the physics and numerics of the modeled process, rather than by programming constraints imposed by the rest of the code. For example, the calculation of diabatic heating due to long-wave radiation is extremely CPU-intensive and has become problematic with the demand for high horizontal resolution within the dynamics. Using a coarser horizontal grid within the long-wave radiation model may significantly reduce CPU time while having little impact on the system accuracy. Results may in fact improve if the reduced CPU time could allow for more frequent radiation calls using more frequent cloud information." [1]

Figure 1 provides the high-level design of the GEMS systems. In the figure, Fortran 90 modules are denoted with circles. Application modules supported by GEMS consist of ocean, dynamics, slow physics and fast physics, and diagnosis (LSM_DIAG and LLS_DIAG) utilities. The GEMS framework provides couplers for those application components to exchange information and clock utilities for controlling the running sequence. In addition, the GEMS framework also provides the utilities such as grid and parallel communication for readily developing application components.

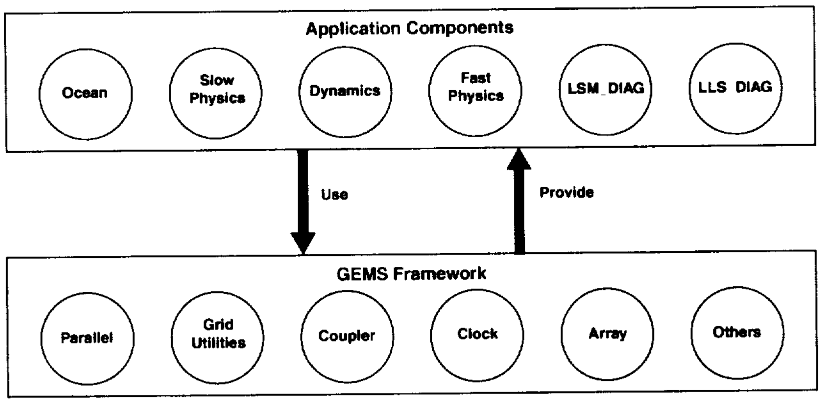

Figure 5. Relationship Between Application and Framework in Aries/NSIPP

The detailed description of the code structure of GEMS can be found in the following:

> "First, GEMS compliancy requires the standardization of model and coupler interfaces, as well as model utilities (e.g., initialize, run, finalize), thus allowing greater ease in writing applications. Memory management within a GEMS model is based on dynamical allocation, and controlled by generic New() and Delete() routines. Interaction between models is channeled through the coupler, which translates quantities from one representation into another. Fortran 90 data structures are used for definitions of the model states and couplers to allow data abstraction and to facilitate multiple instances of the model system within the application. The concept of a class in object-oriented design includes data structures as well as the routines (member functions or methods) which operate on the data structures. In the GEMS framework there are two main classes:

1. **Model class**—As the name suggests, a model class is used to represent the environmental models comprising GEMS. A model class has 3 main data structures:
   - **Input couplings**—Contains the necessary information coming from other models, on the native model grid, to update the model state.
   - **Model state**—Contains the state which is updated by the model.
   - **Output couplings**—Contains the necessary information to force other models, which are not part of the state, on the native model grid.
2. **Hermes Class**—This class contains the coupler utilities to transform from one model representation (grid and domain decomposition) into another. A Hermes class also has 3 main data structures:
   - **Input couplings**—Contains information on a grid of one type and decomposition.
   - **Output couplings**—Representation of input information on a grid of a second type and decomposition.
   - **Transform**—Operators to transform from one grid and domain decomposition to another." [1]

2.5.4 Evaluation

Strengths

GEMS has successfully been applied in climate modeling at NSIPP. Its utilities written in Fortran 90 can be used for developing application codes in Fortran 90. Modular design has been widely used, which reduces the complexity of codes.

Weaknesses

Since Fortran 90 does not have inheritance and polymorphism, the extensibility and flexibility of the GEMS framework is limited. Dynamically allocated storage has been heavily used in GEMS. That is certainly good for efficient use of memory during computing. However, it would be formidable for a code written in C/C++ to communicate with GEMS since the compiler treatment of dynamically allocated storage is different between C/C++ and Fortran 90.

Summary

The GEMS framework, which is written in Fortran 90, satisfies the basic functionality requirements for Earth System Modeling Framework (ESMF) and has been used in NSIPP's production code. However, whether

GEMS can serve as a community framework is worth exploring since a community framework should have good flexibility and extensibility in order to satisfy the needs of various organizations. Two object-oriented concepts, inheritance and polymorphism, are very important in enabling good flexibility and extensibility. Fortran 90 does not have these important features.

2.5.5 References

[1]URL: http://dao.gsfc.nasa.gov/subpages/office-notes.html

2.6 Flux Coupler

2.6.1 Introduction

The Flux Coupler is developed by NCAR [1]. It computes interfacial fluxes between the various component models in the NCAR climate system model (CSM) [2] and distributes these fluxes to all component models while insuring the conservation of fluxed quantities. Currently, the Flux Coupler can support four ocean models (NCOM at NCAR, POP at LANL, regional pacific, and a data-only dummy model), three Atmosphere models (CCM, dummy model I, and dummy model II), and land and ice models.

The Flux Coupler is evolving into a new version (CPl5). The new Flux Coupler will implement the following new features:

- Fewer constraints on surface model domains (e.g., allowing shifted pole grids).
- A more general and flexible method for mapping data between the various grids.
- A more accurate way of taking one atmosphere's net solar flux calculation and applying that to ice, land, and ocean components with widely varying surface albedos.
- Miscellaneous efficiency improvements (e.g., with respect to multitasking).

More detailed information can be found at the ESM Web site [3].

The NCAR CSM is not a particular climate model, but a framework for building and testing various climate models for various applications. In this sense, more than any particular component model, the Flux Coupler defines the high-level design of the CSM software. This report is based on review of documents found at the CSM Web site [1], review of Version CP14 software, and several teleconferences with our POC, Brian Kauffman, and Tom Bettge at NCAR.

2.6.2 Synopsis

Prominent information about the Flux Coupler includes the following:

1. **Community of origin**—Flux Coupler is developed by NCAR for the NCAR CSM.

2. **Description**—It is a separate executable for coupling different climate modeling components such as ocean and atmosphere components.

3. **Team**—Chief developers are Brian Kauffman for Flux Coupler Version CPl4 and Tom Bettge for Flux Coupler Version CPl5.

4. **Maturity**—A few years; CPl4 is the current version and CPl5 is under development.

5. **Users**—There are a few hundred users in the climate modeling community ranging from national laboratories to universities.

6. **Language**—The Flux Coupler source code is written almost entirely in standard Fortran 77. Perhaps the most notable exception is the use of the library calls "msread" and "mswrite" which rely on NCAR's site-specific Mass Storage System (MSS) for storing large output files. Application model components can be written in Fortran 77 and Fortran 90.

7. **Tools/utilities included**—The Flux Coupler has Fortran subroutines to map flux fields between various model grids, combine like fields from several grids onto one grid, and perform summation

and time-average of quantities. However, those subroutines are not sufficiently encapsulated to be used by other components such as the ocean components.

8. **Application interface**—To use the Flux Coupler, the application model components must follow rules set up by the Flux Coupler developers for the data types used in exchanging data between the Flux Coupler and the application component. A user cannot modify the Flux Coupler easily to satisfy his/her needs.

9. **Documentation**—The user's guide for the Flux Coupler is located at [4].

10. **Associated software**—MPI and multitask shared-memory utilities have been used to support parallel communication.

11. **Special features**—In addition to computing and distributing fluxes, the Flux Coupler also controls the execution and time evolution of the complete CSM by controlling the exchange of information between the various components.

2.6.3 Description

The Flux Coupler performs flux computation and exchange for interacting component models within the CSM. This allows the CSM to be broken down into separate components, atmosphere, sea-ice, land, and ocean, that are "plugged into" the Flux Coupler (a.k.a. "driver"). A primary requirement of the Flux Coupler is to ensure that conservative properties—such as momentum, heat, and fresh water—are neither created nor destroyed as they are exchanged between CSM model components. The following characteristics of the Flux Coupler were derived by examining the Flux Coupler source code and documentation:

- The Flux Coupler is a common application for coupling component models in the NCAR CSM. There are four components to a complete system: atmosphere, land, ocean, and sea-ice.
- Each component is required to be connected to the coupler and exchange data through the coupler only.
- Each component model is a separate code with its own computational requirements, such as spatial resolution and time step.
- Individual components can be created, modified, or replaced without necessitating code changes in other components.
- CSM components run as separate executables, communicate via message passing (MPI in particular), and processing can be distributed among several computers.
- The Flux Coupler computes fluxes from component model variables and passes the required fluxes to the components while ensuring the conservation of fluxed quantities.
- The Flux Coupler synchronizes the execution of component models.
- The Flux Coupler performs grid interpolations of data from component models using different grids.
- The Flux Coupler uses parallel communication to exchange data with components.

The high-level design of the Flux Coupler is illustrated in figure 6. The Flux Coupler is written in Fortran 77 and consists of a main program and subroutines. As shown in figure 6, the Flux Coupler provides data exchange services for application models such as the Ocean and Atmosphere components. However, utilities in the Flux Coupler cannot be easily used for developing components because they are not sufficiently encapsulated. The ellipses in figure 6 represent a group of Fortran 77 subroutines. The purpose of those subroutines is to gather, merge, sum, and/or time-average the various component flux fields from various sources and form a set of complete input fluxes for each component model. A nested loop structure is used for controlling the running sequence of components. The ocean model communicates with the coupler once per outer loop iteration, while the atmosphere, ice, and land models communicate once per inner loop iteration. Modular design has been used to facilitate implementation of alternate configurations. A variety of time coordination schemes can be, and have been, implemented by rearranging subroutine calls at the highest level (within the main program file), requiring a minimal amount of code modification or new code.

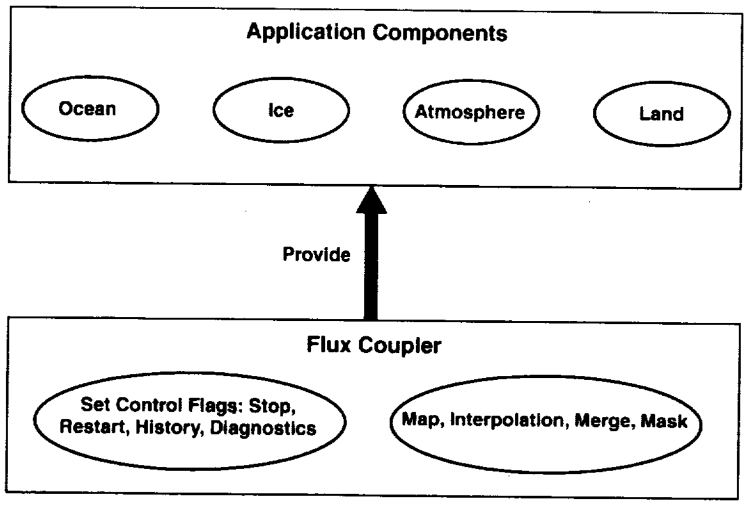
Application Components
Ocean
Ice
Atmosphere
Land
Provide
Flux Coupler
Set Control Flags: Stop, Restart, History, Diagnostics
Map, Interpolation, Merge, Mask

Figure 6. High-Level Design at the Flux Coupler

The Flux Coupler has incorporated a number of subroutines in the areas of mapping and control. The mapping utilities include subroutines for initialization, bilinear interpolation, merging, verifying acceptable component, and masking routines which are listed in table 2. Those subroutines are used in the Flux Coupler; they are not available to the application components.

In the area of control, there are subroutines to set control flags for stopping the calculations, creating restart and history data, and outputting standard diagnostics. These flags are used by the coupler, but are also sent to component models to facilitate the coordination of data sets. The corresponding subroutines are shown in table 3.

Both shared-memory multitasking and message passing (e.g., MPI) have been used in CSM for parallel communication. Inside the Flux Coupler, shared-memory multitasking is used. Between the Flux Coupler and an application component, MPI is used with a wrapper (msg_recv_r, msg_recv_I, msg_send_r, msg_send _I). The application components can use either MPI or shared-memory multitasking.

Since application model components and the Flux Coupler can be run as an individual executable, the load balance among those executables has to be considered. Currently, the load balancing is done manually by trial and error.

2.6.4 Evaluation

Strengths

The Flux Coupler has successfully served the CSM community by providing a utility to plug component models into the CSM. If a user follows the rules for data exchange set up by the Flux Coupler developers, they can test their application components in the environment of CSM through the Flux Coupler without changing the Flux Coupler code; however, their interface is not flexible (see weaknesses).

Weaknesses

The Flux Coupler involves many operations, such as flux calculations and execution control, which makes maintenance and upgrade difficult. One solution is to divide the multiple functionalities into several subroutines and encapsulate the scope of data and functionality in each subroutine.

The high-level design of Flux Coupler is procedural not object-oriented, which limits its extensibility and reusability. The most recent version of the Flux Coupler (CPl5) uses Fortran 90 to modularize the code and some of the flux calculations have been removed, in favor of allowing the individual components to compute the fluxes.

The original design of the Flux Coupler was intended to provide an independent coupling tool allowing users to test their application component in the CSM environment without modifying the Flux Coupler. In reality, users still tend to modify the Flux Coupler code to interface with their application models, rather than simply following the rules set up by the Flux Coupler developers. In general, this procedure can result in significant configuration management issues with the Flux Coupler code. At the current time, most users do not insert new components into the CSM through the Flux Coupler; rather they run simulations with the existing CSM components and modify the parameter sets to tailor the computations to their needs.

Summary

The Flux Coupler can be run as an independent executable providing coupling for application components. It has many users in the climate modeling community. However, its procedural code structure limits the flexibility of its interface for application components. In addition, the Flux Coupler contains several separate functions (exchanging state variables, calculating and distributing fluxes, controlling the execution of a program), which makes the code difficult to maintain and update.

2.6.5 Reference

[1] http://www.cgd.ucar.edu/csm/models/cpl/

[2] http://www.cgd.ucar.edu/csm

[3] http://www.cgd.ucar.edu/csm/models/cpl-ng/

[4] http://www.cgd.ucar.edu/csm/models/cpl/cpl4.0

Table 2. Mapping Utilities

Subroutine Name	Purpose	Assumptions
map_init(Si)	Initialize and/or update any mapping weights or indexes for all mapping options	
map_aavg_x2o_init(ifrac)	Setup routines for area-averaging onto ocean grid	
map_aavg_x2a_init(ifrac)	Setup routines for area averaging onto atmosphere grid. Compute weights for (1) area averaging ice and ocean fields onto the atmosphere grid (2) merging ice, ocean, and land fields (given all fields on atmosphere grid)	Ocean and ice are on identical grids Atmosphere and land are on identical grids The grids used in this subroutine are "edge" (a.k.a. "vertex") grids (i.e., in both the x and y [longitude and latitude] directions), these are the n+1 coordinates of grid cell edges which define the area associated with the n grid cells Any surface area not covered by ocean or ice is covered by land Further assumptions as per routine map_aavg_rsum: -all grids are in degrees (as per spherical/global grids) -all grids are **strictly** increasing -all y grids are in [-90,90] -all x grids are periodic, ie. x(1)+360=x(nx+1) -all grids contain x=180 (minimum requirement: grids overlap)
map_aavg_rsum (x0, y0, nx0, ny0, and x1, y1, nx1, ny1,nc_max, and da0, nc1, i1, j1, da1)	For each 0-grid cell (i,j), this routine computes nc1, i1(n), j1(n), and da1(n), for n=1,nc1, which can then be used to map values (via RIEMAN-UM/ integral) from the 1-grid (which has cell-edge coordinates x1 and y1) onto the 0-grid (which has cell-edge coordinates x0 and y0)	Both grids are in degrees (as per spherical/global grids) All grids are **strictly** increasing Both y grids are in [-90,90] Both x grids are periodic, i.e. x(1)+360=x(nx) The x grids overlap at some point

Table 2. Mapping Utilities (continued)

Subroutine Name	Purpose	Assumptions
map_bilin_init(Xin ,Yin ,mx,my, and Xout,Yout,nx,ny, and w,i1,j1)	Bilinear interpolation routines. For every pair of output grid indices (i,j) compute input grid indices i1(i),j1(j), and four weights w(i,j,1),w(i,j,2),w(i,j,3),w(i,j,4), such that, Fout(i,j) = F(Xout(i),Yout(j)) = w(i,j,1)*Fin(i1 ,j1) + w(i,j,2)*Fin(i1+1,j1) + w(i,j,3)*Fin(i1+1,j1+1) + w(i,j,4)*Fin(i1 ,j1+1) is the bilinear interpolationof field Fin, on grid Xin, Yin, onto field Fout, on grid Xout,Yout	All coordinates (x,y) lie in [0,360]x[-90,90] (i.e., lie on the globe, with units of degrees) X and Y grids are strictly increasing X(1:mx) is periodic, so that Xin(mx+1) is understood to be Xin(1,j)+360.0, likewise, Fin(mx+1,j) is understood to be Fin(1,j)if Y(1) > -90 and/or Y(ny) < +90, Yin(0,j) is understood to be -90.0 Yin(my+1,j) is understood to be +90.0 and corresponding north- and south-pole values may need to be fabricated by the routine that uses these weights and indices, is such cases, Fin(i,0) is understood to be a fabricated south-pole value, and Fin(i,my+1) is understood to be a fabricated north-pole value
merge3(nx,ny, w1,w2,w3, f1,f2,f3, fsum)	Merges three quantities on the same grid, using associated weights to obtain a net quantity on the grid. Presumably the weights add up to 1.0	
merge2(nx,ny, w1,w2, f1,f2, fsum)	Merges two quantities on the same grid, using associated weights to obtain a net quantity on the grid. Presumably the weights add up to 1.0	
map_check()	Verify acceptable component model domains	atmosphere and land are on identical grids ice and ocean are on identical grids ice and ocean have identical domains
map_maskl()	set (or reset) mask_l such that mask_l(i,j) is non-zero iff the coupler will use land data at location (i,j) Note: wm_l2a(i,j) > 0 => coupler will use land data at (i,j)	atmosphere and land are on identical grids

Table 3. Control Utilities

Subroutine Name	Purpose	Assumptions
control_diag()	Set control flags for the creation of runtime diagnostics	This routine is called once at every time step
control_hist()	Set control flags for the creation and archiving of history files	This routine is called once at every time step
control_rest()	Set control flags for restart file creation	
control_stop()	Set control flags for when the integration stops	

2.7 Distributed Data Broker

2.7.1 Introduction

The University of California, Los Angeles (UCLA) Distributed Data Broker (DDB) [1] results from a collaboration between the Department of Atmospheric Sciences [2] at UCLA and the Computer Science Department [3] at the University of California, Berkeley (UCB). The DDB, as described on the DDB homepage, is a

> "software tool to handle distributed data exchanges between ESM (Earth Science Model) components. The DDB differs from conventional coupler tools in that it handles communication between two independent ESM components without the need for a central communication agent. The DDB has three major components: the Model Communication Library (MCL), the Communication Library (CL) and the Data Translation Library (DTL). The MCL contains a set of callable routines that are used by the different ESM components to register during an ESM run, and perform the exchanges of data. The CL is a set of routines used by the DDB to manage the data exchanges based on the communication libraries supported by the available computer platforms. Lastly, the DTL transforms data in a given grid domain to the domain of the requesting model. This library will include a number of utilities from simple linear interpolation routines to high-order data translation functions. The Distributed Data Broker works in a producer-consumer paradigm in which the data producers send the data directly to the consumers at given time intervals. The data consumers will later receive the data at a rate dictated by its internal computations." [1]

2.7.2 Synopsis

Prominent information about DDB includes the following:

1. **Community of origin**—DDB originates in the Atmospheric Science Department at UCLA and the Computer Science Department at UCB.

2. **Description**—DDB provides a communication mechanism between climate models, enabling them to share gridded information without an intermediate agent process. Instead it provides for direct communication between components without a separate process.

3. **Team**—Past and present members of the DDB team include Tony Drummond [4], C. Roberto Mechoso [5], J.A. Spahr, James Demmel [6], Howard Robinson [7], and Keith Sklower [8]. Tony Drummond, the POC, has recently left UCLA [2] for a new position at NERSC [9]. The DDB team is in the process of deciding how the data broker project will be supported. Tony is also a member of the Common Component Architecture Forum [10] and the Common Modeling Infrastructure Working Group (CMIWG) [11].

4. **Maturity**—This project began 1994.

5. **Users**—DDB is primarily used within UCLA for coupling climate models in the context of the Earth System Model (ESM) [12]. It has been used by the National Partnership for Advanced Computational Infrastructure (NPACI) [13] to perform multiscale multiresolution modeling [14] using Legion [15]. There is only one user outside the UCLA group.

6. **Language**—DDB is written in C++ and C. The library provides interfaces in both Fortran and C, thus allowing it to work with models written in multiple languages.

7. **Tools/utilities included**—DDB comes packaged as three libraries:
    - the Model Communication Library (MCL);
    - the Communication Library (CL);
    - the Data Translation Library (DTL).
8. **Application interface**—DDB employs a simple interface scheme in which models register at the beginning of the task using a registration broker and they communicate during the task using send or receive subroutine calls. The architecture for this system also seems to be influenced by Common Object Request Broker Architecture (CORBA) [16] [17].
9. **Documentation**—There are several documents specifically about DDB including the home page [1], a recent presentation [18] by Mechoso and others [19]. Much of the information about DDB is derived from information about applications which use it, the primary one being the ESM [12] at UCLA, for which the DDB plays a prominent role [20]. Other publications include articles on parallelization and performance of atmospheric chemical tracer models by Demmel and Smith [21] [22], the UCLA Atmospheric General Circulation Model (AGCM) [23] by Mechoso et al. [24] [25], coupling to an ocean model by Farrara et al. [26], and others [27] [28]. Some fragmentary documentation is also available over the Web for project status and results [11]. There is no user/reference manual. Mechoso notes ([18], p. 19) that DDB will soon be available upon request from UCLA or at the U.S. National HPCC Software Exchange (NHSE) [29].
10. **Associated software**—The DDB can use either PVM [30] or MPI [31] libraries.
11. **Performance**—The DDB is designed to replace an equivalent service which runs as a separate centralized computational process on an independent processor. In actual tests, as shown in figure 10, DDB does indeed perform faster than such a system.
12. **Special features**—DDB is targeted towards highly distributed processing. It can work in conjunction with Legion [15], a distributed virtual computing system developed at the University of Virginia.

2.7.3 Description

DDB is designed to operate in a multimodel system where each model distributes its processing across multiple processors according to a geographic grid. DDB provides the functionality of an intermediate communication agent without the implementation of an intermediate communication agent.

High-Level Design

DDB consists of three libraries—an MCL, a CL, and a DTL. It is designed to serve as a communications interface between multiple components in a simulation.

The Problem DDB Is Designed to Avoid: Centralized Coupling

DDB is designed to solve problems associated with centralized coupling approaches where, on an N processor system with M models, N-1 processors are dedicated to model processing and one is designated as a communication agent. On such a system, as M (and N) increases, there is an increasing processing burden which is cast upon the single communication process. Thus the potential exists to have N-1 processors waiting while an overloaded single processor handles all of the communication between them. This situation can be aggravated in cases where different models have different grid sizes where a geographic region on one model (corresponding to a single processor) would be, for a different model with a different grid, associated with multiple processors that all own corresponding parts of the subgrid. In such a situation, coupling data from a region on one model to another model might involve numerous processors, all of which can be affected by a communication bottleneck. Thus, this method of dividing the processors into model and communication processor tasks has performance risks.

The DDB Solution: Distributed Coupling

The DDB approach provides the appearance of a central coupler with the implementation of a distributed coupler. In this case on an N processor system all N processors can be dedicated to model processing and when communication is required, it is handled on a point-to-point basis with both processors sharing the communication processing burden. Thus this is a solution which is more appropriate for distributed processing and can scale more effectively. DDB can be shown diagrammatically as an intermediate agent between all models but functionally it is implemented as a function call which operates in each processor's process space.

DDB Implementation

The DDB provides the functionality of a communication agent between model processes without the agent being implemented as a separate process. Figure 7 illustrates how DDB is considered in the context of the ESM [12], incorporating the following models:

- the UCLA AGCM [23];
- the Oceanic General Circulation Model (OGCM)[32];
- the UCLA Atmospheric Chemical Model (ACM) [33].

The models communicate using DDB as an intermediate agent but DDB does not run as a separate task. The diagram shows all models using function call API, implemented as part of the DDB library.

Schematic Representation of the UCLA Earth System Model

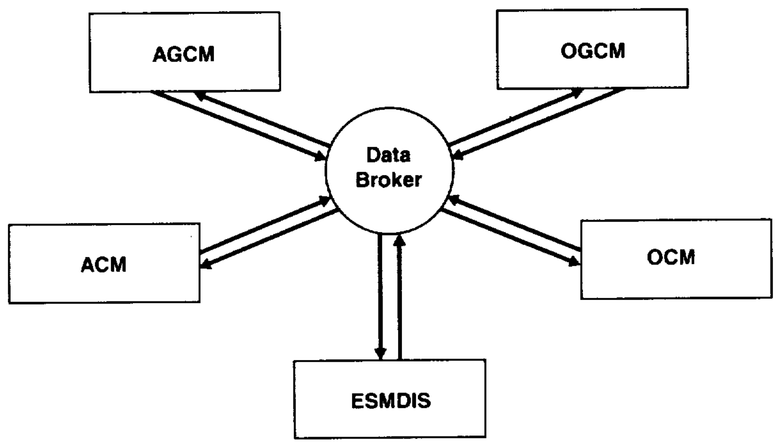

Figure 7. DDB as Part of the ESM

As described on the DDB home page, the DDB operates in two phases. In the first phase, it employs a registration broker to collect model information. In the second phase, it functions as a communication intermediary between the other models. Each of these phases are explained below.

Model Registration

To use the DDB each process must register with it at the beginning of each run. A Registration Broker (RB), implemented as subroutine calls, is used to collect information on each model which includes both the information it can supply and the information it requires from other models. This information specifically incorporates information about grids. The RB returns to the model a set of information which can be used in connection with the later communication calls. After registration, each process has enough information to communicate with the other models. This procedure is illustrated in figure 8.

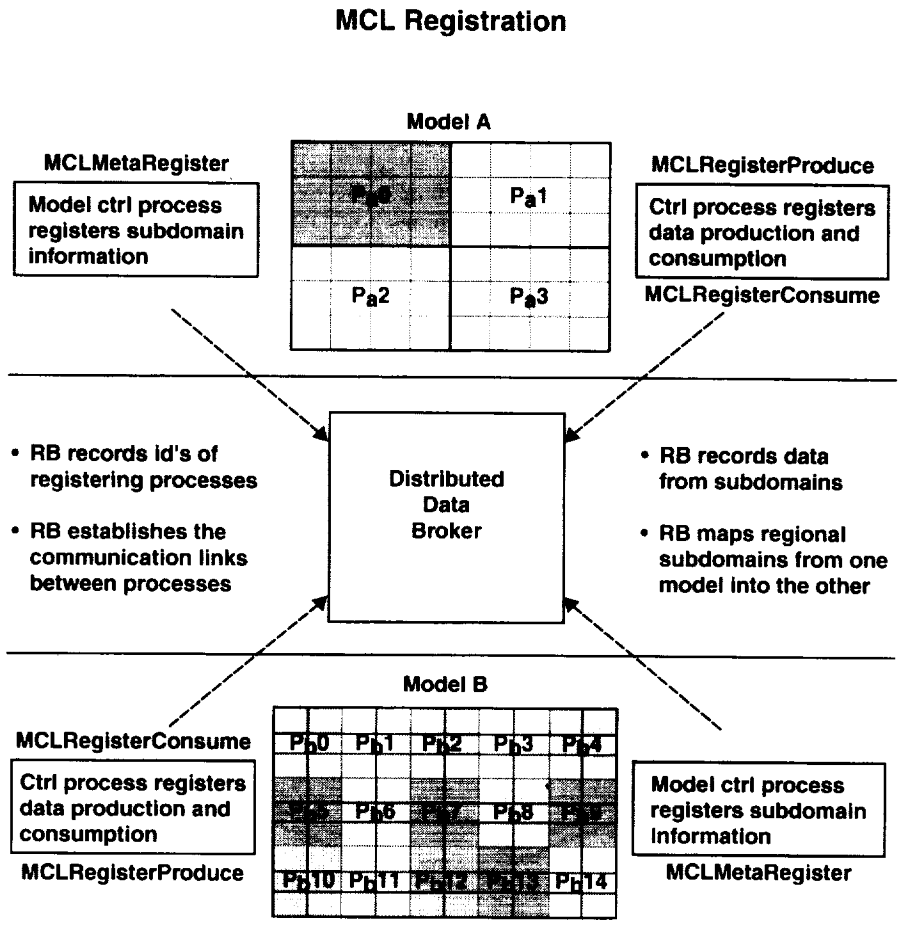
MCL Registration
Model A
MCLMetaRegister
Model ctrl process registers subdomain information
Pa0
Pa1
Pa2
Pa3
MCLRegisterProduce
Ctrl process registers data production and consumption
MCLRegisterConsume
• RB records id's of registering processes
• RB establishes the communication links between processes
Distributed Data Broker
• RB records data from subdomains
• RB maps regional subdomains from one model into the other
Model B
MCLRegisterConsume
Ctrl process registers data production and consumption
MCLRegisterProduce
Pb0
Pb1
Pb2
Pb3
Pb4
Pb5
Pb6
Pb7
Pb8
Pb9
Pb10
Pb11
Pb12
Pb13
Pb14
Model ctrl process registers subdomain information
MCLMetaRegister

Figure 8. DDB Registration [1]

Inter-Model Communication

During the rest of the run, models communicate using MCLGetData and MCLSendData function calls as illustrated in figure 9. As described by Drummond:

> "In figure 9, Pa0 requests data from Model B. The region in the Model B's domain that corresponds to the subdomain being worked by Pa0 is highlighted by the orange rectangle.

This rectangle covers the subdomains being worked by Pb0, Pb1, Pb2, Pb5, Pb6, Pb7. Therefore, a simple call to MCLGetData is translated in to 6 receives-operations from each of the Model B processes. Likewise, a single MCLSendData from each of the Model B processes will get translated into one or more send-operations to Model A processes. After all the data is received by Pa0, it pastes the different information received and translates the data to its own grid units." [1]

The communication takes place above the MPI [31] level. Each send may be translated into multiple MPI calls.

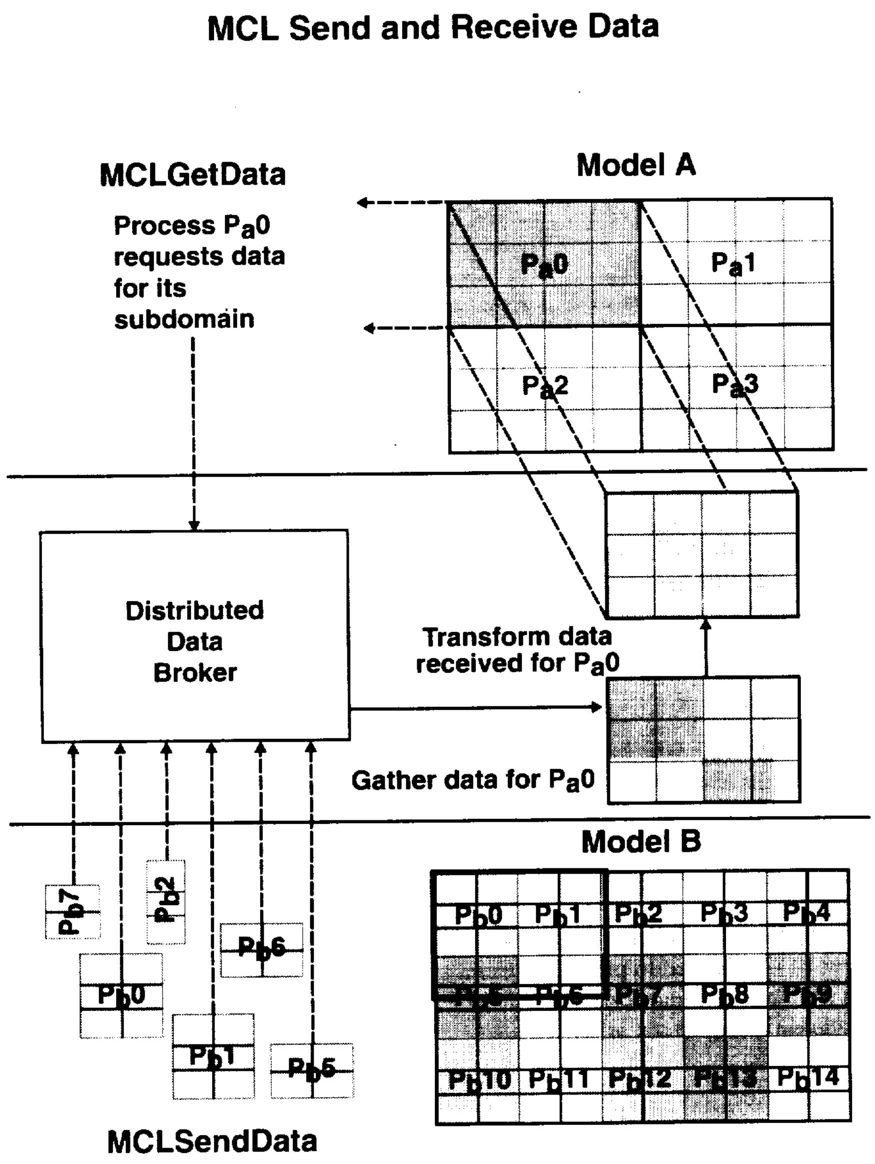
MCL Send and Receive Data
MCLGetData
Process P_a0 requests data for its subdomain
Model A
P_a0
P_a1
P_a2
P_a3
Distributed Data Broker
Transform data received for P_a0
Gather data for P_a0
Model B
P_b7
P_b2
P_b6
P_b0
P_b1
P_b5
P_b0 P_b1 P_b2 P_b3 P_b4
P_b5 P_b6 P_b7 P_b8 P_b9
P_b10 P_b11 P_b12 P_b13 P_b14
MCLSendData

Figure 9. Inter-Model Communication [1]

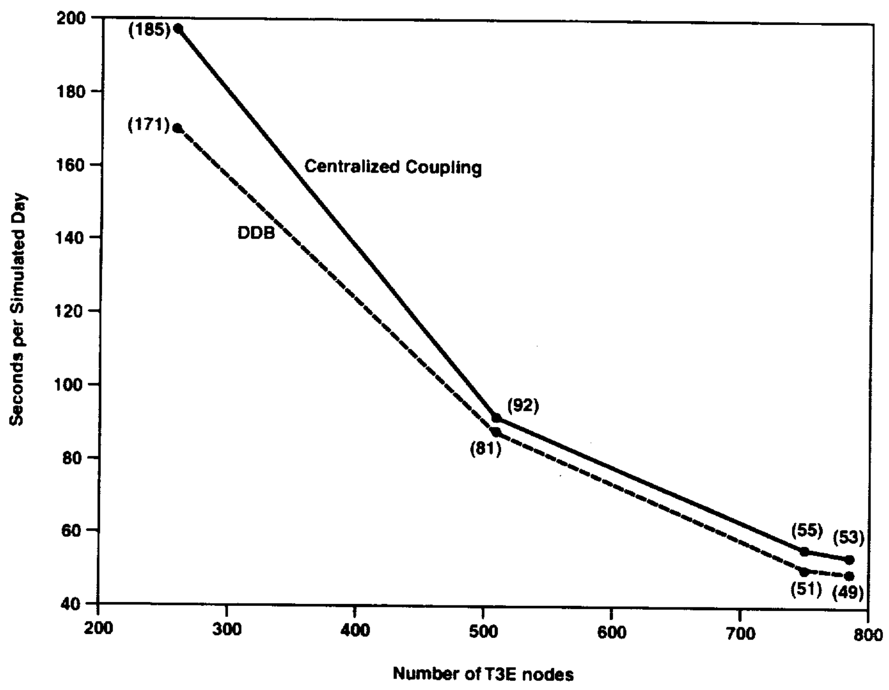

Figure 10. Comparison of Centralized Coupling Versus DDB Coupling as a Function of Number of T3E Nodes [1]

Performance

Performance results for DDB versus a centralized coupling implementation are shown in figure 10. The figure shows a decrease in time (seconds per simulation day) ranging from 7.5 percent for 250 nodes to 12 percent for 500 nodes and 7.5 percent for 800 nodes.

2.7.4 Evaluation

According to Tony Drummond, who has investigated other coupling software similar to DDB and presented concise comparison results [34] to CMIWG [11], the DDB has characteristics in common with the NCAR Flux Coupler [35], CSU Flux Coupler [36], CERFACS Coupler (OASIS) [37] [38], and the Max Suarez Coupler [39] in that it provides a modular interface between different numerical models. DDB is unique in its registration and distributed coupling approach.

According to the definition, DDB is not a framework as "a framework is a reusable, semi-complete application that can be specialized to produce custom applications," ([40], p. 4), but it has the potential to play an important role in any framework in that it efficiently "describes the interface of each object and the flow of control between them," ([40], p. 4).

Strengths

The strengths of the DDB are the following:

1. **Library**—It is packaged as a library, which means it can be easily incorporated into other framework solutions.

2. **Language**—It has dual Fortran and C interfaces which eases integration into larger frameworks.
3. **Simple API**—The API for registering models and communicating between them is simple, a sign of good design.
4. **Distributed**—DDB can work in distributed environments without tying up another processor.
5. **Registration**—The DDB uses a registration process for the models, a procedure which should be incorporated into any framework.
6. **Proof of concept**—DDB shows that this concept works.

Weaknesses

The weaknesses of DDB are as follows:

1. **Few users**—It has a small user community and therefore may not be as robust as other couplers with more users. OASIS [38], on the other hand, has at least 15 user groups as of 1997.
2. **Lack of documentation**—The lack of a user manual, reference manual, or other documentation is a serious deficiency, particularly compared to other couplers which provide User's Guides, such as OASIS [41] and NCAR FC [42].
3. **Uncertain support**—The DDB team does not seem to offer the level of support of Cactus [43] or ROOT [43]. DDB seems mainly to be an internal project. The issue of support becomes more uncertain with the loss of Tony Drummond [4] to NERSC [9].

Summary

The UCLA DDB is not, by itself, a complete framework. Nevertheless, the task it seeks to accomplish, exchange of gridded data, represents a crucial part of any framework solution. DDB seeks to fill the role of coupler using a distributed technique. This technique works and provides some performance gains in comparison to centralized coupling, which makes it worthy of close examination for the underlying technology because the general design concept is excellent. At the same time the number of users is small, there is little specific documentation and the level of future support from the team appears uncertain.

2.7.5 References

[1] DDB. URL: http://www.atmos.ucla.edu/~drummond/DDB

[2] UCLA Department of Atmospheric Sciences (UCLA_DAS). URL: http://www.atmos.ucla.edu

[3] UCB Department of Computer Science (UCB_CS). URL: http://www.cs.berkeley.edu

[4] Tony Drummond, 510-486-7624, LADrummond@lbl.gov. URL: http://www.atmos.ucla.edu/~drummond

[5] C. Roberto Mechoso, 310-825-3057, mechoso@atmos.ucla.edu.
URL: http://www.atmos.ucla.edu/~mechoso/

[6] James Demmel, 510-643-5386, demmel@cs.berkely.edu. URL: http://www.cs.berkely.edu/~demmel/

[7] Howard Robinson, 510-642-4979, hbr@cs.berkeley.edu. URL: http://www.cs.berkeley.edu/~hbr/

[8] Keith Sklower, 510-642-9587, sklower@cs.berkely.edu. URL: http://www.cs.berkely.edu/~sklower/

[9] NERSC. URL: http://hpcf.nersc.gov/

[10] DOE Common Component Architecture Project (DOE_CCA).
URL: http://www.extreme.indiana.edu/~gannon/cca_report.html

[11] CMIWG. URL: http://janus.gsfc.nasa.gov/~mkistler/infra/master.html

[12] Earth System Model (UCLA_DAS_ESM). URL: http://www.atmos.ucla.edu/esm/

[13] NPACI. URL: http://www.npaci.edu/index.html

[14] Multi-Scale Multi-Resolution Modeling (NPACI_MSMRM).
URL: http://www.npaci.edu/Research/ESS/projects/msmr.html

[15] Legion (UVA_Legion). URL: http://www.cs.virginia.edu/~legion/

[16] Common Object Request Broker Architecture (CORBA). URL: http://industry.ebi.ac.uk/~corba/

[17] Group, Object Management, 1991: The Common Object Request Broker - Architecture and Specification. Object Management Group.

[18] Mechoso, C.R., 2000: A Distributed Data Broker for Multi-Disciplinary Applications.
URL: http://www.atmos.ucla.edu/~mechoso/toulouse_workshop/

[19] Drummond, Leroy, Carlos Mechoso, J. Demmel, J. Robinson and K. Sklower, 1999: A Distributed Data Broker for Coupling Multiscale and Multiresolution Applications, 1999 Advanced Simulation Technologies Conference High Performance Computing Symposium, Apr 11-15, SCS.
URL: http://www.scs.org/confernc/astc99/html/hpc-pp.html

[20] Farrara, John, Leroy Drummond, Carlos Mechoso and A. Spahr, 1999: An Earth System Model for MPP Environments: Performance Optimization and Issues in Coupling Model Components, 1999 Advanced Simulation Technologies Conference High Performance Computing Symposium.
URL: http://www.scs.org/confernc/astc99/html/hpc-pp.html

[21] Demmel, J. and S.L. Smith, 1994: Parallelizing a Global Atmospheric Chemical Tracer Model, Proceedings of the Scalable High Performacne Computing Conference, Knoxville, TN, May 23-25, IEEE Computer Society, 718-725.

[22] Demmel, J. and S.L. Smith, 1995: Performance of a Parallel Global Atmospheric Chemical Tracer Model, Proceedings of Supercomputing '95, IEEE Computer Society.

[23] AGCM. URL: http://www.atmos.ucla.edu/esm/agcmdir/

[24] Mechoso, C.R., C.C. Ma, J.D. Farrara, J.A. Spahr and R.W. Moore, 1993: Parallelization and Distribution of a Coupled Atmosphere-Ocean General Circulation Model. Monthly Weather Review, 121, 2062-2076.

[25] Mechoso, C.R., L.A. Drummond, J.D. Farrara and J.A. Spahr, 1998: The UCLA AGCM in High Performance Computing Environments, Supercomputing '98.
URL: http://www.atmos.ucla.edu/~drummond/SC98_1

[26] Farrara, John D., Leroy A. Drummond, Carlos R. Mechoso and Joseph A. Spahr, 1998: An Atmospheric General Circulation Model with Chemistry for the CRAY T3E – Design Performance Optimization and Coupling to an Ocean Model, Proceedings from Second International Workshop on Software Engineering and Code Design in Parallel Meteorological and Oceanographic Applications, 251-264.

[27] Wehner, M.F., A.A. Mirin, P.G. Eltgroth, W.P. Dannevik, C.R. Mechoso, J.D. Farrara and J.A. Spahr, 1995: Performance of Distributed Memory Finite Difference Atmospheric General Circulation Model. Parallel Computing, 21, 1655-1675.

[28] Wehner, M.F., J.J. Ambrosiano, J.C. Brown, W.P. Dannevik, P.G. Eltgroth, A.A. Mirin, J.F. Farrara, C.C. Ma, C.R. Mechoso and J.A. Spahr, 1993: Towards a High Performance Distributed Memory Climate Model, Second International Symposium on High Performance Distributed Computing, Spokane Washington, IEEE Computer Society, 102-113.

[29] NHSE. URL: http://www.nhse.org

[30] PVM. URL: http://www.epm.ornl.gov/pvm/pvm_home.html

[31] MPI. URL: http://www-unix.mcs.anl.gov/mpi/

[32] OGCM. URL: http://www.atmos.ucla.edu/esm/ogcmdir/

[33] Atmospheric Chemical Model (ACM). URL: http://www.atmos.ucla.edu/esm/acmdir

[34] Drummond, Tony, 1998: A Partial List of Available Coupling Software, Infrastructure Working Group Meeting, Tucson, AZ, Oct 15-16. URL: http://janus.gsfc.nasa.gov/~mkistler/infra/docdir/csw.html

[35] Flux Coupler Version 4.0 (FC 4.0). URL: http://www.cgd.ucar.edu/csm/models/cpl

[36] CSU Flux Coupler (CSU FC).
URL: http://kiwi.atmos.colostate.edu/BUGS/groupPIX/don/fc/fluxcoupler.html

[37] European Centre for Research and Advanced Training in Scientific Computation (CERFACS).
URL: http://www.cerfacs.fr

[38] OASIS Flux Coupler (OASIS). URL: http://www.cerfacs.fr/globc/software/oasis/oasis.html

[39] Max Suarez, 301-614-5292, max.suarez@gsfc.nasa.gov.

[40] Fayad, Mohamed E., Douglas C. Schmidt and Ralph E. Johnson, 1999: Building Application Frameworks. Wiley.
URL: http://www.amazon.com/exec/obidos/ASIN/0471248754/qid%3D968778251/102-0715276-9734544

[41] Valcke, Sophie, Laurent Terray and Andrea Pacentini, 2000: OASIS 2.4: Ocean Atmosphere Sea Ice Soil User's Guide, CERFACS. URL: http://www.cerfacs.fr/globc/software/oasis/doc_oasis2.4.ps

[42] Kauffman, Brian G., 1998: The NCAR CSM Flux Coupler Version 4.0 User's Guide, NCAR.
URL: http://www.cgd.ucar.edu/csm/models/cpl/cpl4.0/doc0.html

[43] Cactus . URL: http://www.cactuscode.org

2.8 Other Frameworks

This section examines three other frameworks—ROOT, PAWS, and ALICE— in somewhat less detail than the prior six.

2.8.1 ROOT

2.8.1.1 Introduction

ROOT arises from the European Organization for Nuclear Research (CERN) [1]. ROOT is described in the Executive Summary of the mission statement on the ROOT home page as follows:

> "The ROOT system provides a set of OO frameworks with all the functionality needed to handle and analyze large amounts of data in a very efficient way. Having the data defined as a set of objects, specialized storage methods are used to get direct access to the separate attributes of the selected objects, without having to touch the bulk of the data. Included are histogramming methods in 1, 2 and 3 dimensions, curve fitting, function evaluation, minimization, graphics and visualization classes to allow the easy setup of an analysis system that can query and process the data interactively or in batch mode.
>
> "Thanks to the built-in CINT C++ interpreter the command language, the scripting, or macro, language and the programming language are all C++. The interpreter allows for fast prototyping of the macros since it removes the time-consuming compile/link cycle. It also provides a good environment to learn C++. If more performance is needed the interactively developed macros can be compiled using a C++ compiler.
>
> "The system has been designed in such a way that it can query its databases in parallel on MPP machines or on clusters of workstations or high-end PC's. ROOT is an open system that can be dynamically extended by linking external libraries. This makes ROOT a premier platform on which to build data acquisition, simulation and data analysis systems." [2]

2.8.1.2 Synopsis

Prominent information about ROOT includes the following:

1. **Community of origin**—ROOT arises from CERN [1] in the particle physics community. It was designed to support data analysis for the Large Hadron Collider where the expected amount of data produced exceeded 1 million GB (1 PB) per year.

2. **Description**—"ROOT is a system for large-scale data analysis and data mining. It is being developed for the analysis of particle physics data, but it can be equally well used in other fields where large amounts of data need to be processed." [3]

3. **Team**—Team members include Rene Brun [4] and Fons Rademakers [5].

4. **Maturity**—this project has been functioning since 1994 and is still active. Version 2.25/02 has recently been released.

5. **Users**—According to a paper by Rademakers and Brun in 1998, "More than 16,000 copies of the ROOT binaries have been downloaded from [the] Web site, about 700 people have registered as ROOT users, and the Web site gets more than 150,000 hits per month. ROOT is currently being used in many different fields: physics, astronomy, biology, genetics, finance, insurance, pharmaceutics,

etc." [3] Since that time the number of distributed binaries has increased to more than 75,000. The ROOT home page also lists at least 37 applications using ROOT [6]. Thus, ROOT is a broadly disseminated development framework.

6. **Language**—C++

7. **Tools/utilities included**—According to Brun [7], class categories include basic ROOT classes, container classes, histogram and minimization classes, tree and n-tuple classes, two-dimensional graphics classes, three-dimensional and detector geometry classes, Graphical User Interface (GUI) classes, interactive interface classes, the C++ interpreter [8], the operating system interface, networking classes, and documentation classes.

8. **Application interface**—The application interface consists of the object-oriented class structures. The interfaces are well thought-out and extremely well documented.

9. **Documentation**—The ROOT Web site has an enormous amount of documentation including 51 tutorials [9], 37 applications examples [6], a reference guide [10], and numerous other documents. An interesting presentation by Federico Carminati describes the migration by CERN to C++/OO away from Fortran [11]. On page 6, he discusses how this choice was made:

   - Migrate immediately to C++.
   - Adopt the ROOT framework.
   - Allow use of Fortran and C++.
   - Impose a single framework.

   He also discusses specific policies adopted regarding programming styles in C++ (p. 15), support for a Bazaar [12] model of development (p. 16), and lessons learned in migrating scientists familiar with Fortran to a new environment (p. 25).

10. **Associated software**—The C++ interpreter [8] is an associated software package that has been tightly integrated into the ROOT system.

11. **Performance**—Unknown.

12. **Special features**—According to the developers the main features of ROOT include the runtime type information system [13], the object I/O system [14], and automatic documentation generation [15]:

   - The runtime type information system [13] is a memory resident dictionary that, at runtime, maintains information on all objects including lists of all global functions, all global variables, all classes, data member descriptions of the classes, and class member functions. This centralized information source is a powerful capability not implemented in any other framework surveyed.
   - The object I/O system [14] includes a set of classes to support I/O to/from machine independent files. It is "designed to be particularly efficient for objects manipulated by physicists: histograms, n-tuples, trees, and events."
   - ROOT provides for automatic documentation generation [15] using a set of documentation classes which "allows the creation of hyperized (in HTML format) C++ header and source files, inheritance trees, class indices, macros, and session transcripts. Thanks to this facility almost everything in the ROOT system can be automatically documented and cross-referenced." This is a tremendously useful capability not duplicated in any other framework.

- ROOT provides an excellent example of an object hierarchy. It has 310 classes grouped in about 24 frameworks divided into 14 categories [7].
- ROOT uses a C interpreter (CINT) [8], for the scripting language. This is a great example of how a scripting language should be associated with a framework, though it is not necessarily obvious that interpreted C or C++ represents the best choice.

2.8.1.3 Description

ROOT is primarily oriented towards data analysis instead of simulation. The distribution in focus between event generation, detector simulation, event reconstruction, data acquisition, and data analysis is shown in figure 11.

Unlike many frameworks, ROOT has a true class inheritance hierarchy. The ROOT schema organization is loosely illustrated in figure 12. Basic classes in an inheritance hierarchy are shown in figure 13.

Figure 14 illustrates the ROOT environment and tools.

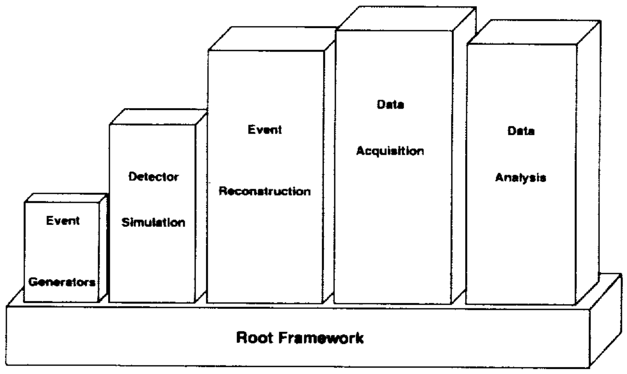

Figure 11. The Primary Speciality of the ROOT Framework is Event Reconstruction, Data Acquisition, and Analysis [2]

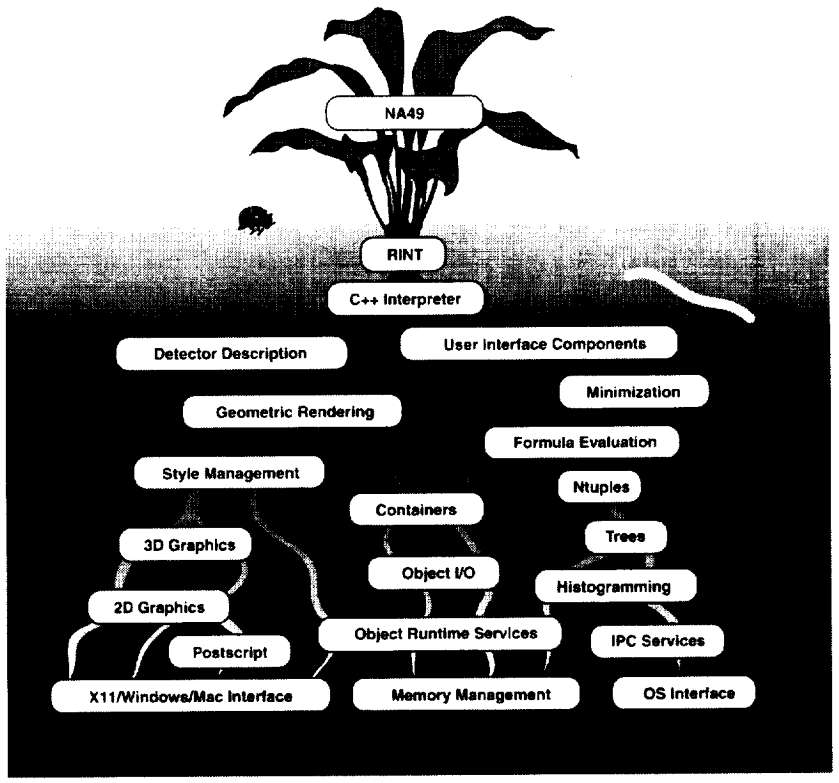

Figure 12. ROOT Schema [2]

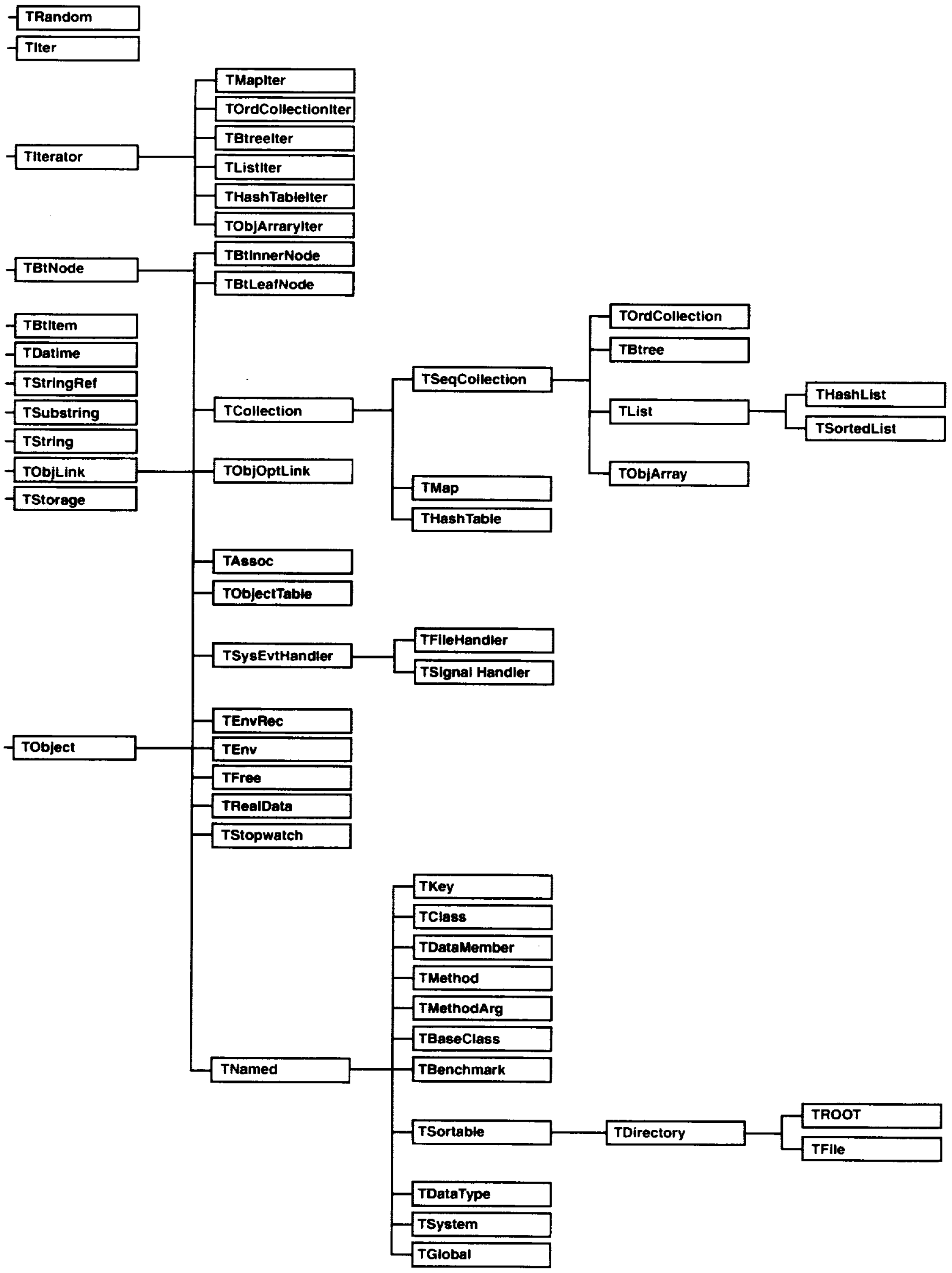

Figure 13. Basic ROOT Classes in an INheritance Hierarchy [2]

ROOT Environment and Tools

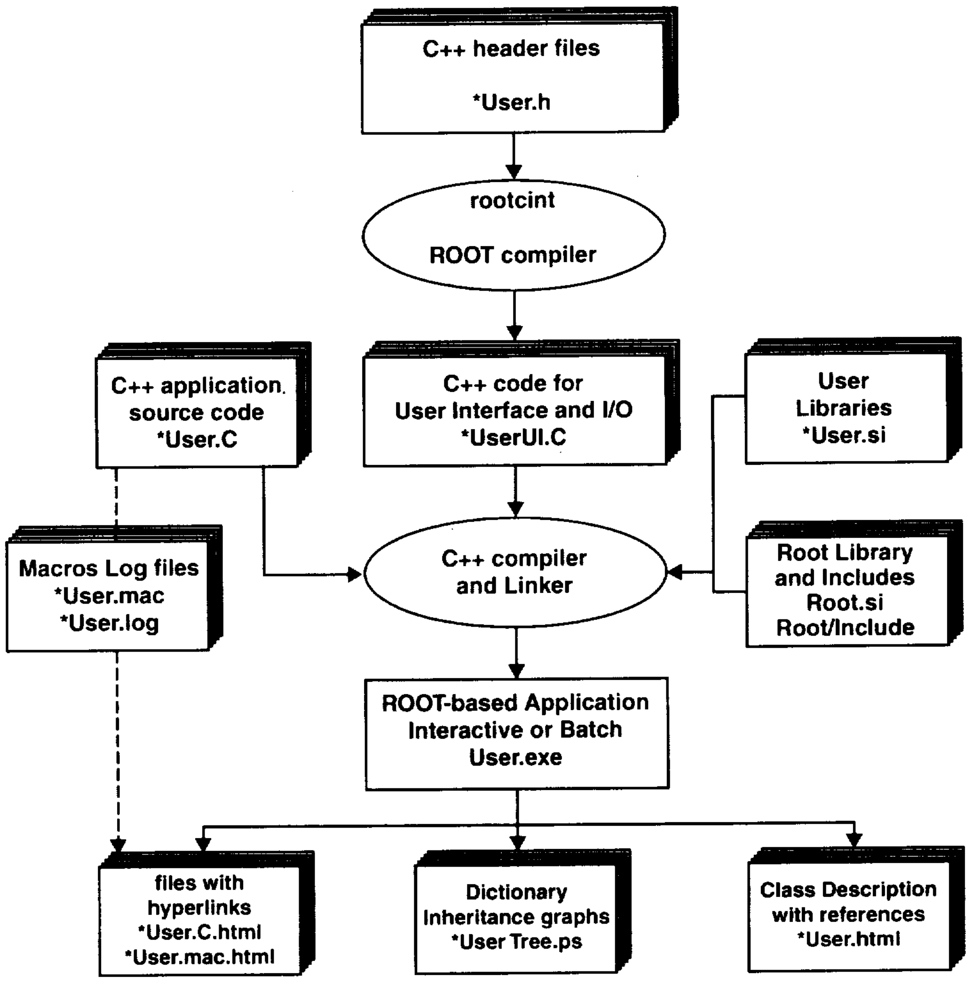

Figure 14. ROOT Environment and Tools [2]

2.8.1.4 Evaluation

ROOT is an excellent example of a scientific object-oriented framework. At the current time it is not specifically suited to handling models for the climate community.

Summary

ROOT exceeds all other surveyed frameworks in terms of number of users, maturity, extent of documentation, and excellence in object-oriented design. Is worthy of close inspection and emulation by the climate community. The fact that a scientific organization such as CERN has migrated from Fortran to C++ using ROOT provides lessons regarding options available to the climate community.

2.81.5 References

[1] European Organization for Nuclear Research (CERN). URL: http://www.web.cern.ch/CERN/

[2] ROOT (ROOT). URL: http://root.cern.ch/Welcome.html

[3] Rademakers, Fans and Rene Brun, 1998: ROOT – An Object-Oriented Data Analysis Framework. Linux Journal, 51.

[4] Rene Brun, +41 22 76 74124, Rene.Brun@cern.ch. URL: http://consult.cern.ch/xwho/people/00526

[5] Fons Rademakers, Fons.Rademakers@cern.ch. URL: http://root.cern.ch/~rdm/

[6] Applications Using ROOT. URL: http://root.cern.ch/root/ExApplications.html

[7] Brun, Rene and Fons Rademakers, 1996: ROOT Architectural Overview. URL: http://root.cern.ch/root/Arhitecture.html

[8] C/C++ Interpreter (CINT). URL: http://root.cern.ch/root/Cint.html

[9] ROOT Tutorials. URL: http://root.cern.ch/root/Tutorials.html

[10] Brun, Rene and Fons Rademakers, 2000: ROOT Reference Guide. URL: http://root.cern.ch/root/Reference.html

[11] Carminati, Federico, 2000: ALICE AliRoot Framework. URL: http://root.cern.ch/cgi-bin/print_hit_bold.pl/root/R2000Html/AliRoot/img0.htm

[12] Raymond, Eric S., 1999: The Cathedral and the Bazaar. URL: http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html

[13] Rademakers, Fons, 1996: The ROOT Dictionary. URL: http://root.cern.ch/root/Dictionary.html

[14] Brun, Rene and Fons Rademakers, 1996: the ROOT Object I/O System. URL: http://root.cern.ch/root/InputOutput.html

[15] Brun, Rene, Nenad Buncic and Fons Rademakers, 1997: ROOT Automatic Documentation Generation. URL: http://root.cern.ch/root/Documentation.html

2.8.2 PAWS

2.8.2.1 Introduction

The PAWS originates in the Advanced Computing Laboratory (ACL) [1] at LANL [2] with some collaboration from the NEXUS project [3] at ANL [4] and the POOMA project [5] at LANL [2]. As described on the home page:

> "PAWS (Parallel Application Workspace) is a software infrastructure for use in connecting separate parallel applications within a component-like model. 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 API to indicate which data structures are to be shared and at what points the data are ready to be sent or received. PAWS implements a general parallel data descriptor, and automatically carries out parallel layout remapping when necessary. Connections can be dynamically established and dropped, and can use multiple data transfer pathways between applications. PAWS uses the NEXUS [3] communication library and is independent of the application's parallel communication mechanism." [6]

2.8.2.2 Synopsis

Prominent information about PAWS includes the following:

1. **Community of origin**—PAWS originates at ACL [1].
2. **Description**—Provides a mechanism to connect separate parallel applications.
3. **Team**—Team members include Peter Beckman [7], Patricia Fasel [8], Bill Humphery, Sue Mniszewski, and Teresa Roberts.
4. **Maturity**—This package does not seem to be very mature. There are few papers and few users.
5. **Users**—There are no known users outside of the PAWS team.
6. **Language**—C++
7. **Tools/utilities included**—PAWS is considered an ACTS [9] toolkit component.
8. **Application interface**—PAWS has a Fortran and C interface.
9. **Documentation**—There are several documents specifically about PAWS including a users guide [10], a programmers manual [11], a programmers reference [12], a conference publication [13], and an overview presentation [14]. The home page also provides a project overview and technical summary.
10. **Associated software**—PAWS employs the NEXUS [3] communication mechanism for its message passing substrate, and is currently in the process of adding the ability for PAWS applications to work within the GLOBUS [15] metacomputing environment. Initial test applications use the POOMA [5] framework. The POOMA framework includes an interface that uses PAWS to allow all POOMA multidimensional array objects to be shared with other programs.

11. **Performance**—Performance has not been evaluated.

12. **Special Features**—TBD

2.8.2.3 Description

The PAWS framework concept has something in common with the role played by the UCLA DDB [16] and the NCAR Flux Coupler [17]: the sharing of information between running modules. PAWS is primarily concerned with the sharing of scalars, multidimensional fixed-size arrays, and multidimensional varying-sized arrays. Like the flux coupler, PAWS has a separate controller process. Unlike the flux coupler, it has a scripting language based on Tcl [18].

Figure 15 shows functional segments and interfaces of the PAWS framework. Figure 16 shows the relationship between the controller and other applications.

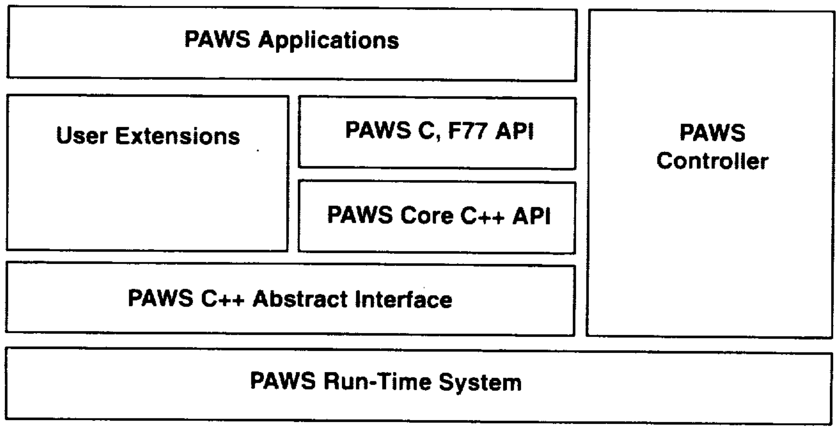

Figure 15. Breakdown of PAWS Programming Interface and Controller [10]

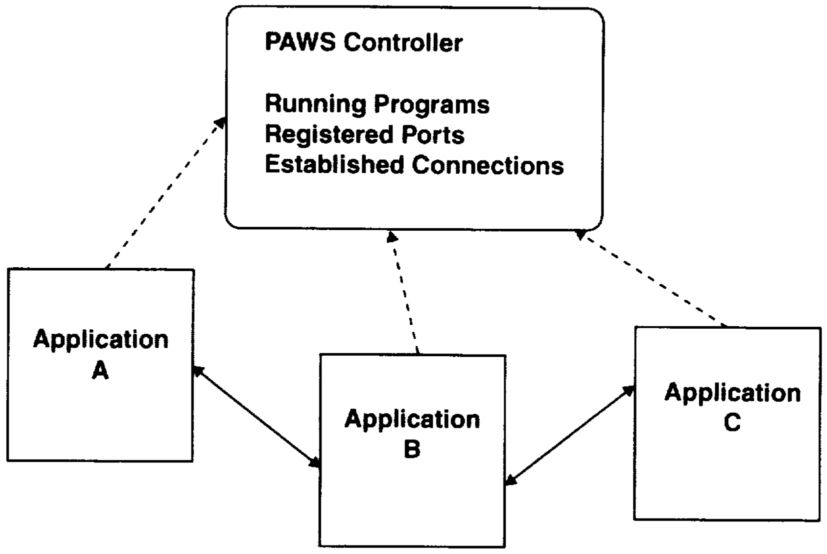

Figure 16. The PAWS Controller Interacting with Several Applications [10]

2.8.2.4 Evaluation

PAWS does not perform interpolation functions or mapping to coordinate systems like the Flux Coupler or data broker. For this reason it is probably not suited for specific tasks needed by the climate community.

Summary

PAWS does not seem mature enough at this time to merit a great deal of consideration by the climate community. Nevertheless, the concepts of having a controller associated with a scripting language and that of registration of the applications with the controller, are useful examples of capabilities that a framework should have.

2.8.2.5 References

[1] ACL. URL: http://www.acl.lanl.gov

[2] LANL. URL: http://www.lanl.gov/worldview

[3] NEXUS. URL: http://www.globus.org/nexus

[4] ANL. URL: http://www.anl.gov

[5] POOMA. URL: http://www.acl.lanl.gov/pooma/

[6] PAWS. URL: http://www.acl.lanl.gov/paws/

[7] Peter Beckman, beckman@acl.lanl.gov.

[8] Patricia Fasel, 505-667-3533, pkf@lanl.gov. URL: http://www.c3.lanl.gov/~pkf/

[9] ACTS Toolkit (ACTS). URL: http://www.nersc.gov/ACTS

[10] Beckman, Pete, Pat Fasel, Bill Humphrey, Sue Mniszewski and Teresa Roberts, 2000: PAWS User's Guide, Version 1.3. URL: http://www.acl.lanl.gov/paws/docs/UserMan/index.html

[11] Beckman, Pete, Pat Fasel, Bill Humphrey, Sue Mniszewski and Teresa Roberts, 2000: PAWS Programmers's Manual, Version 1.3. URL: http://www.acl.lanl.gov/paws/docs/ProgMan/index.html

[12] Beckman, Pete, Pat Fasel, Bill Humphrey, Sue Mniszewski and Teresa Roberts, 2000: PAWS Reference Manual, Version 1.3. URL: http://www.acl.lanl.gov/paws/docs/ProgRef/index.html

[13] Beckman, Peter H., Patricia K. Fasel and William F. Humphrey, 1998: Efficient Coupling of Parallel Applications Using PAWS, *High Performance Distributed Computing (HPDC) 7, Chicago, IL.*

[14] Humphrey, William, 1998: Efficient Coupling of Parallel Scientific Applications Using PAWS. URL: http://www.acl.lanl.gov/paws/papers/slides_PawsSummary/

[15] GLOBUS. URL: http://www.globus.org

[16] DDB. URL: http://www.atmos.ucla.edu/~drummond/DDB

[17] Flux Coupler Version 4.0 (FC 4.0). URL: http://www.cgd.ucar.edu/csm/models/cpl

[18] Tcl/Tk (TclTk). URL: http://dev.scriptics.com/

2.8.3 ALICE

2.8.3.1 Introduction

Advanced Large-Scale Integrated Computational Environment (ALICE) originates in the Mathematics and Computer Science Division of ANL [1]. As described on the homepage:

"The goal of the ALICE (Advanced Large-Scale Integrated Computational Environment) project is to eliminate barriers in using independently developed software components in the construction of high-performance numerical applications. We believe this will lay the groundwork for widespread exploitation of teraflop-scale computational resources and for new scientific insights.

"The ALICE project, a collaborative effort among researchers in the Mathematics and Computer Science Division of Argonne National Laboratory, has grown out of our long tradition of expertise in high-performance software. This experience has demonstrated the benefits of encapsulating numerical and parallel computing expertise in user-ready tools. However, the complexity of today's large-scale scientific simulations often necessitates the combined use of multiple software packages to address areas such as mesh manipulations, numerical solution of partial differential equations, optimization, sensitivity analysis, and visualization. While efficient and robust tools exist, combining them remains difficult because of data management and interoperability problems. ALICE research focuses on

1. developing low-overhead mechanisms for integrating extensible software for scientific problem solving; and,
2. building component-based toolkits that encapsulate expert knowledge in numerical algorithms and parallel computing.

"ALICE development is motivated by a range of large-scale scientific applications that ensure the relevance and practicality of our design. Our approach supports both new and legacy applications, thereby enabling scientists to reuse legacy kernels and to program in the style of most comfort to them, for example, traditional Fortran development or more object-oriented paradigms. In addition, we are actively engaged in dialogues within the DOE Common Component Architecture Forum [2] concerning component-based software interoperability throughout the DOE high-performance computing community." [3]

2.8.3.2 Synopsis

Prominent information about ALICE includes the following:

1. **Community of origin**—ALICE originates at ANL [1].

2. **Description**—A set of tools for building scientific applications.

3. **Team**—Primary investigators include Ibrahima Ba, Satish Balay [4], Steve Benson, Anthony Chan, Paul Fischer, Lori Freitag, Bill Gropp [5], Paul Hovland, Jeff Linderoth, Rusty Lusk, Lois Curfman McInnes, Jorge Mor, Lucas Roh, Barry Smith, Deb Swider, Rajeev Thakur, Henry Tufo, Steve Wright, and Golbon Zakeri.

4. **Maturity**—As a whole ALICE does not seem mature because of sparse documentation. Individually, the tools may be just fine.

5. **Users** – projects using ALICE include
   - The Center on Astrophysical Thermonuclear Flashes (ASCI ASAP Center)
   - Multi-Model Multi-Domain Computational Methods in Aerodynamics and Acoustics (NSF Multidisciplinary Challenge Project)
   - Massive Crystallographic and Microtomographic Structural Problems (DOE Grand Challenge Project)

6. **Language**—ALICE has a multilanguage architecture of object-oriented libraries that are usable by Fortran, C, C++, and Java.

7. **Tools/utilities included**—The core of the ALICE infrastructure is formed by the following tools:
   - Automatic Differentiation: ADIC and ADIFOR—ADIFOR and ADIC are source translators that augment Fortran 77 and C programs with derivative computations.
   - High-performance portable I/O: ROMIO—A high performance, potable implementation of MPI-IO.
   - Message-passing tools: MPICH [6]—A specification for the user interface to message passing libraries for parallel computers.
   - Optimization: MINPACK and NEOS—Libraries of advanced optimization software.
   - PDE and numerical linear algebra software: PETSC [7] and BlockSolve95—Function libraries.
   - Unstructured Mesh Computations: SUMAA3D [8]—A framework for parallel unstructured mesh computations.
   - Computational Steering—a new project to develop a computational steering system based on hierarchical adaptive analysis, parallel computing, and immersive visualization.
   - Futures Lab: [9]—Explores, develops, and prototypes next-generation computing and communications infrastructure systems.
   - Distributed Supercomputing Tools: GLOBUS [10].

8. **Application Interface**—With respect to interfacing, the ALICE home page [3] provides the following information:

   "Critical is the design of interfaces to handle the interconnections among components. Clearly, no single mechanism will solve all problems. ALICE uses multiple layers to provide both coarse-grained functionality in connecting applications together and fine-grained functionality for high-performance data sharing. Key design features are:

   - a numerical object interface based on mathematical abstractions (e.g., the interface specifies classes of linear and nonlinear solvers as opposed to particular algorithms and data structures), and
   - a common interface specification among components that may use various underlying implementations to address portability and performance issues."

9. **Documentation**—The sole document for ALICE is an overview slide presentation [11]. Individual documentation exists for some of the components and is available from the ALICE homepage [3].

10. **Associated software**—The whole framework is a collection of tools.

11. **Performance**—Performance can only be evaluated for individual tools.

12. **Special features**—This framework is oriented towards interoperability between a diverse collection of tools.

2.8.3.3 Description

Most of the information available on ALICE is conceptual and high level, though some detailed information is available for the individual components. Figure 17 shows the ALICE infrastructure concept where ALICE components are used by applications and in turn are based on lower-level components. Figure 18 illustrates the type of components that can be potentially integrated into the ALICE framework. Figure 19 illustrates the relationship between ALICE components and toolkits. Figure 20 illustrates the ALICE vision of how components would work together in advanced applications.

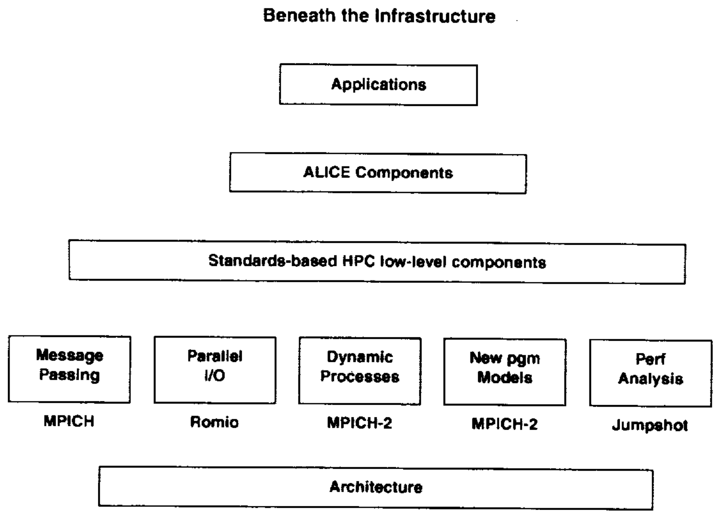

Figure 17. ALICE Infrastructure Concept (from [11], p. 15)

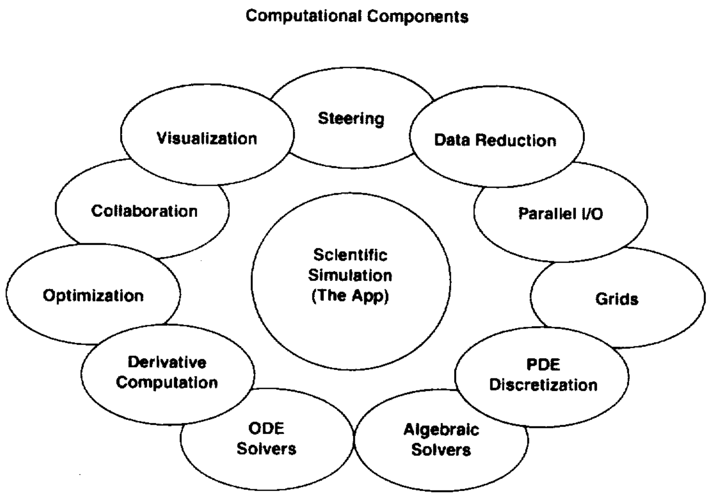

Figure 18. ALICE Concept of Computational Components (from [11], p. 8)

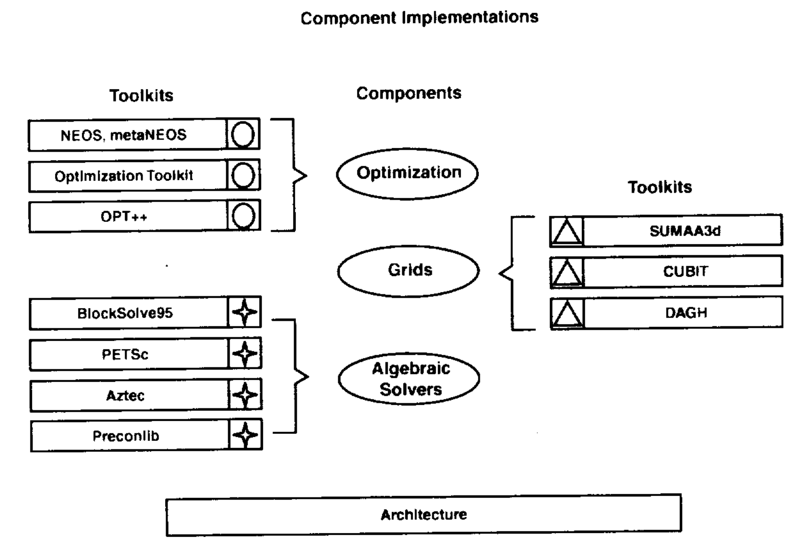

Figure 19. ALICE Component Implementations (from [11], p. 10)

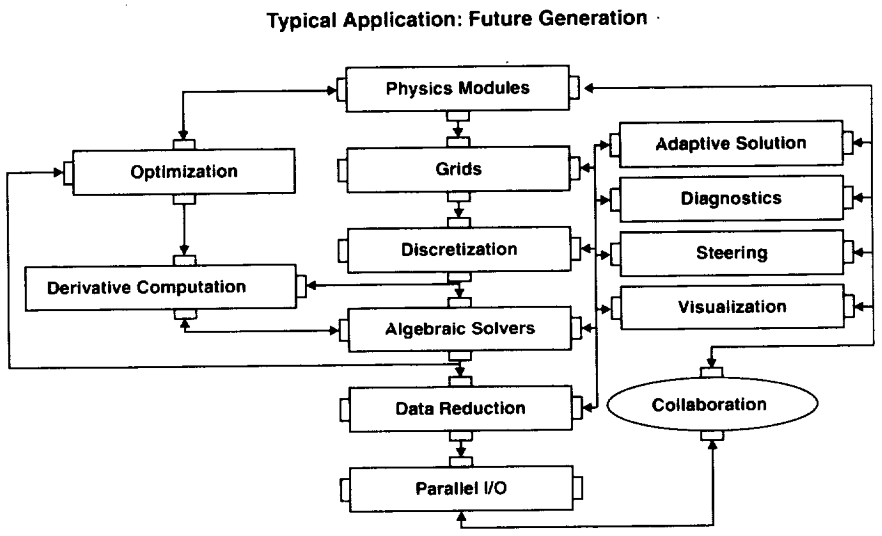

Figure 20. ALICE Vision of Next Generation Applications (from [11], p. 12)

2.8.3.4 Evaluation

There is not sufficient information on ALICE to make a useful evaluation. Based on the amount of available information, this project does not seem to have matured yet.

Summary

ALICE is useful in that it has developed a high-level concept of how a variety of components could work together in a framework. More information is required before this framework could be recommended for the climate community.

2.8.3.5 References

[1] ANL. URL: http://www.anl.gov

[2] Common Component Architecture Toolkit (CCAT).
URL: http://www.extreme.indiana.edu/ccat/index.html

[3] ALICE. URL: http://www-unix.mcs.anl.gov/alice/

[4] Satish Balay, balay@mcs.anl.gov. URL: http://www-fp.mcs.anl.gov/~balay

[5] William Gropp, gropp@mcs.anl.gov. URL: http://www-unix.mcs.anl.gov/~gropp/

[6] Message Passing Interface (MPICH). URL: http://www-unix.mcs.anl.gov/mpi/mpich

[7] Portable Extensible Toolkit for Scientific Computing (PETSC). URL: http://www.mcs.anl.gov/petsc/

[8] Scalable Unstructured Mesh Algorithms and Applications (SUMAA3d).
URL: http://www-unix.mcs.anl.gov/sumaa3d/

[9] Futures Lab (ANL_FuturesLab). URL: http://www-fp.mcs.anl.gov/fl/

[10] GLOBUS. URL: http://www.globus.org

[11] Staff, DOE, 1998: What is ANL Thinking About?
URL: http://www-unix.mcs.anl.gov/alice/presentations/doe2000_retreat98/index.htm

3.0 Discussion and Recommendations

3.1 Motivation for Earth Science Modeling Framework (ESMF)

The primary motivation behind the ESMF [1] is to provide a community software infrastructure, in the spirit of the report [1] issued by the Common Modeling Infrastructure Working Group (CMIWG) [2] and in the spirit of the Cooperative Agreement Notice (CAN) [3], and provide an infrastructure which

1. provides a common software base for the entire community;
2. specifies systems and methods by which models are developed but not the models themselves;
3. focuses more upon the interactions and collaborations ([4], p. 4) between the models and less upon the implementation of the models;
4. provides efficient and flexible means of communication and coupling between models;
5. provides the means, including tools and utilities, to develop focused core models [1] which can be used as standard models within the community;
6. provides sufficient flexibility to allow the community to develop variations and alternatives of core models as needed for research, experimentation, and operations;
7. is designed so that modeling advances can be primarily determined by community consensus and scientific merit instead of software favoritism or rigidity or the disposition of the organization responsible for supporting the framework;
8. can support both operational and research modeling efforts;
9. provides technology-transfer mechanisms to allow research model capabilities to migrate to the operational centers;
10. caters specifically to a community skilled in the Fortran language, recognizing and supporting Fortran as a valued language for representing functional mathematical relationships;
11. provides a means to not only reuse code that has been developed in the past, but to also reuse code that will be developed in the future;
12. provides a bridge to new computing technologies, allowing researchers to access new capabilities without requiring major modifications to established and trusted code;
13. fosters documentation of individual models, communication between researchers, and cooperation among organizations.

The ESMF does all this with lower costs, shorter development times, more simplicity, higher reliability, and faster speeds, even as it is used with more complex computing architectures and with more models written by more participants from more diverse backgrounds to solve increasingly difficult problems.

3.2 Current Status of Existing Surveyed Frameworks

In the previously established context of what an ideal framework should provide, the surveyed frameworks can be grouped into several categories:

1. Couplers for Earth science application components.
2. Community level application frameworks.
3. Object-oriented frameworks/toolkits.
4. Earth science frameworks.

3.2.1 Couplers for Earth Science Application Components

A Coupler Is Not Really a Framework, but Plays a Crucial Role

Couplers do not have the breadth to be considered as framework by themselves. Nevertheless, by providing crucial information coupling services between models, couplers are found near the core of any framework design concept because they represent the primary collaboration and communication mechanism between the models and their limitations and strengths primarily determine what can and cannot be done within the framework. An inappropriately chosen coupling mechanism can force users outside the framework to obtain either speed or enhanced communication, thus defeating the framework concept in its entirety, by promoting the proliferation of additional interfaces within the community. On the other hand, a strategically chosen coupler can relieve researchers of communication difficulties and allow them to focus on more scientifically diverse problems, thus promoting the use of the framework in the spirit of community collaboration.

Two Couplers Considered: DDB and Flux Coupler

Two couplers considered in this survey include the UCLA Distributed Data Broker (DDB) [5] and the National Center for Atmospheric Research Flux Coupler [6] as only two of the many [7] couplers that could potentially be considered.

DDB has a Better Design

Of the two couplers, DDB clearly stands out as being designed upon better principles than the Flux Coupler. DDB is primarily designed as a coupler and only a coupler, whereas the Flux Coupler design incorporates coupling, flux computation and control into a single unit which constitutes a defect with serious long term architectural consequences for the entire framework.

Why the Flux Coupler Design Is Flawed

To understand why the FC design is flawed it is important to properly understand the role of a framework, such as in the context discussed by Fayad and Johnson ([4], p. 4). A framework provides at least two major services:

1. It describes the interface between objects.
2. It provides a capability known as *inversion of control* (p. 5) which is a supervisory service, at a high level, for all of the participant objects in the simulation.

Both of these capabilities are needed in any major simulation framework. It is understandable, in the absence of a true simulation framework and because the functions are somewhat related, that the designers of the Flux Coupler would combine them into a single unit. Notwithstanding this choice, the issues of communication between participant models and control of the same models are two separate functions and should be properly be designed and implemented as two different capabilities in separate modules: a coupling capability and a control capability.

The combination of these two distinct capabilities in a single unit, which in addition to combining the two separate principles also executes as a single and separate computing process, is a choice that not only complicates the coupling process but also works against the implementation of a true high-level simulation controller, either by the framework developers or by individual researchers.

If used in a community framework, the Flux Coupler, as a separate entity combining control and communication functions, has the potential to be subject to continued modification and upgrades as researchers require either improved control, improved communication, or additional coupling with new and different models. Because of these requirements, the Flux Coupler will have to undergo continual changes as researchers from the community work closely with NCAR to get it to incorporate new capabilities to talk to new models they wish to use or old ones they wish to modify. Each change may impact the interface, which in turn may impact every member of the community who has implemented a model using that interface. This may in turn drive the researchers to find ways of communicating between models without using the Flux Coupler, so they are not subject to a changing interface for a crucial service. This will in turn defeat the entire framework concept.

These, then, are the potential long-term consequences resulting from the flaw in the Flux Coupler design which combines communication with control in a single module.

DDB Does Not Have the Flux Coupler Flaws

The DDB, on the other hand, has none of the Flux Coupler flaws. It is designed primarily as a general-purpose coupler upon sound principles similar to those used by Common Object Request Broker Architecture (CORBA) [8], an object-oriented communication architecture. Furthermore, it does not run as a separate process but executes in the process space of the communicating models, which is an advantage for distributed system because it reduces bottlenecks. The interface is simple: Models register at the beginning and communicate with each other directly as needed using straightforward calls. DDB is implemented as three libraries and has C and Fortran interfaces, thus it can integrate nicely with almost any framework architecture.

The Interface Is the Most Crucial Part of a Framework

For a function as crucial to a framework as communication and coupling between participant objects, nothing is more important than a simple, flexible, and stable interface. It does not matter, hypothetically speaking, if DDB runs slower, has fewer interpolation options, supports fewer message-passing libraries, does not support as many grid systems, and is scientifically inferior in every way to Flux Coupler or any other coupling system.

In such cases, DDB performance could be improved, interpolation options could be added, more message passing libraries could be supported, as could additional grid systems, and so on. In each and every case these changes would hardly impact the interface. This, then, is the most important feature because it promotes the use of the community framework.

Disadvantages Associated with DDB

DDB is developed by the University of California, Los Angeles (UCLA) group and has only one user outside the group. Tony Drummond, the lead developer, has left the group for a new position at NERSC. The issue of long-term support is currently uncertain. The DDB documentation is sparse. DDB does not have the benefit of having been used by many users, so it may potentially have bugs. These are all negatives which are completely unrelated to the strength of the DDB design but which need to be overcome.

Advantages Associated with Flux Coupler

Flux Coupler is available from NCAR, a larger organization with a Web site containing a large amount of high-quality documentation. Flux Coupler is in its fourth release and is probably quite stable and has more users than DDB. These are all positives that, unfortunately, cannot compensate for the weakness of the Flux Coupler design.

3.2.2 Community Level Application Frameworks

Two of the Contenders: Cactus and ROOT

There are two major open source frameworks which qualify fully as true general-purpose application frameworks: Cactus [9] and ROOT [10].

ROOT Is Elegant and Is in C++

ROOT is an elegant example of the kinds of services that can be provided by a true object-oriented framework. The elements of the framework are well designed and, as a result of the common inheritance tree, have high consistency that provides numerous capabilities with simple interfaces. One of the strongest points is a fine documentation system, a critical component for any scientific community. ROOT has users numbering in the thousands. It is written in C++. It has almost no climate-specific features. These would all have to be added by the climate community as well as a means to effectively integrate Fortran modules into a C++ framework. Whether or not ROOT is actually used in an eventual climate framework, its design should be carefully studied so that as many features as possible can be imitated.

Cactus Is Practical for a Fortran Community

Cactus is a community level framework but has different strengths than ROOT. Cactus supplies object-oriented features like inheritance by simultaneously attaching them on top of both Fortran and C. This is no small feat, considering the differences between the two languages. Cactus accomplishes this task, though with an awkwardness which is inherent to this type of approach.

Cactus provides a high-level development environment with numerous interfaces to Fortran 77, Fortran 90, C, and numerous other systems. It can produce executables that can run on laptops or parallel computers. It carefully defines the interface by the control structure (flesh) and modules (thorns) and implements communication between modules effectively. It also provides powerful supervisory capabilities, such as inspecting a running simulation via a Web browser [11]. This is an enticing example of a type of new service which can be effectively implemented by powerful frameworks such as ROOT or Cactus without requiring changes to legacy code. Cactus has many users, a strong support group, is mature and stable, has a lot of documentation, but no climate-specific support features.

If the Climate Community Is Not Willing to Migrate to C++ then Cactus Is a Better Choice

In a contest between Cactus and ROOT for choice of framework for the climate community, Cactus would clearly be superior because of its explicit support for Fortran modules. Nevertheless, climate models would have to be modified at the highest levels to conform to the Cactus interface. Using Cactus requires the full commitment of the community as once the code is modified, at a high-level, for the Cactus interface, it cannot be independently compiled with a Fortran compiler as it was before. It is not clear that the Cactus interface would work directly for coupling between models in a manner similar to DDB or Flux Coupler. It is likely that some cooperation with the Cactus team would be necessary to incorporate these types of features. The Cactus team has a good track record of cooperating with many institutions.

3.2.3 Object-Oriented Frameworks/Toolkits

Two C++ Contenders: Parallel Object-Oriented Methods and Applications (POOMA) and Overture

POOMA [12] and Overture [13] are two frameworks which are written in C++ and are used for Partial Differential Equations (PDE) problems. Both frameworks offer a user the opportunity to build an application using components from the framework: a collection of C++ classes. Recognizing that migrating to C++ from Fortran is a difficult task, each of them takes different approaches.

POOMA Uses Expression Templates to Achieve Speed Gains

POOMA takes the approach of using the standard C++ STL libraries and implementing high-speed classes using an expression template technique [14] [15] [16]. Regardless of the speed benefits, the benefits are achieved by the introduction of a type of complexity that will impede researchers migrating to C++ from Fortran. This technique is also subject to larger compile times due to the use of a technique that coerces the compiler into performing optimizations for which it is not ideally designed.

The POOMA approach was used to support Parallel Application Workspace (PAWS) [17], another framework for connecting parallel applications that was briefly considered in this survey. The POOMA team has virtually disbanded recently and future support is questionable, although it is likely that their sponsor will continue to need POOMA capabilities. This fact, in addition to the complexity of the approach, suggests that this framework should not be considered for inclusion in a climate community framework.

Overture Uses a Preprocessor to Achieve Speed Gains

Like POOMA, Overture seeks to obtain speed improvements in C++. Overture uses a different approach than POOMA. First, it developed a powerful array package, A++/P++ [18], which is available separately and may be of interest to the climate community. Second, building on these classes it implements a variety of functions. Third, it uses a source-to-source translator, SAGE++ [19] [20], to improve execution speed instead of expression templates. This is a more straightforward procedure though it introduces an extra step into compilation. Thus, independent of Overture itself, there are at least two components of Overture, A++/P++ and SAGE++, which can be of use to the climate community framework. A disadvantage of Overture is that the team has their own research work and cannot as easily support independent requests for software changes, as could the POOMA team (before it lost most of its people).

Overture Approach Is More Practical

In either case, the strict use of C++ and the current situations with both POOMA and Overture make it unlikely that the climate community can adopt either framework completely, though parts of Overture could be used. These issues are still separate from the Fortran/C++ speed issue in which C++ can still stand improvement. The importance of this issue needs to be addressed separately by the climate community given that Fortran is faster but C++ is better designed for constructing frameworks, possibly requiring source-code translators to merge the two should a joint approach be taken.

3.2.4 Earth Science Frameworks

Two Contenders: Goddard Earth Modeling System (GEMS) and Flexible Modeling System (FMS)

Two frameworks arising directly in the Earth science community are GEMS and FMS [21]. (Note: GFDL's FMS was not investigated as part of this Task 4 activity, but was examined in the Task 2 survey of current Earth system modeling applications, as was GEMS.) These frameworks have neither the elegance of ROOT, the full-service development features of Cactus, nor the object-oriented characteristics of POOMA or Overture. What they do have is a direct orientation towards the primary function of the climate community: climate modeling.

Both Are Specifically Geared to Climate Issues

Both GEMS and FMS are Fortran 90 implementations and both address model coupling issues. FMS has an excellent amount of documentation and GEMS has little. Both are currently used for modeling problems. Both of these frameworks have an inherent weakness in that they do not use a true object-oriented language.

3.3 Summary

This section contains overall observations coming out of the survey as well as opinions regarding ESMF development.

Observations:

- No single existing framework can be used without change to construct the ESMF. All frameworks incorporate *some* desirable features; all lack *some* desirable features.
- The majority of the climate community code is written in Fortran 77/90. On the other hand, Fortran is not extensively used outside the research community. Most researchers do not want to migrate to other languages (e.g., C++). The scientific community questions C++ performance relative to Fortran.
- The most powerful frameworks identified (ROOT and Overture) are written in C++. No equivalently powerful frameworks are written in Fortran. There are, however, many useful Fortran libraries and utilities.
- Fortran is missing two key OO features that are available in C++ and Java: inheritance and polymorphism. Fortran 2000 is scheduled to have these features in a few years.
- C++ can be combined with Fortran in only a limited way. There is no standard way for C++ programs to read dynamically-allocated Fortran 90 data structures. This problem will still exist with Fortran 2000.

Opinions:

- OO design and algorithmic design are complements to one another, not competitors. Some system components are better expressed with algorithms. Other system components are better expressed with objects.
- Most complex systems incorporate complementary components. Microsoft Excel is both data visible (spreadsheet) and algorithm visible (Visual Basic Macros). The Overture framework has a high-level OO structure but some of its linear solvers are written in Fortran.

- Physics focuses on algorithmic expressions, which are best represented by a functional language such as Fortran. Frameworks focus on relationships between participating objects (models), which are best represented by an OO language, such as C++, Java, or Smalltalk.
- It may be that in the climate community, the issue of language, instead of evolving properly into a question regarding the appropriate blend of complements (OO <u>and</u> algorithmic expression, Java <u>and</u> Fortran, etc.) has evolved into a competition between complements (Fortran <u>or</u> C++, etc.)
- *The issue of language is absolutely critical to the design of the ESMF.*
- If a community ignores the complementary relationship and tries to design a framework using a functional language (Fortran) instead of an OO language, it will incur complexity problems in the complementary space. For example, POOMA's goal is to achieve Fortran performance with an OO language (C++). It uses expression templates to make compiler optimizations for which it was not designed. The result is extreme complexity. On the other hand, if a framework is designed using Fortran, the framework will be less capable, more complex, and will have less flexibility in the interfaces.
- OO design is critical for building a flexible and extensible ESMF meeting the CAN requirements. Inheritance and polymorphism are the important OO features that are building blocks of a *comprehensive* framework. Inheritance is a way of distributing capabilities to models that minimizes the programming burden on the researchers. Polymorphism provides a simple mechanism for researchers to modify models. Together, inheritance and polymorphism reduce cost and error. They provide a clear hierarchical structure for a complex problem.
- Properties of a framework are derived from the capabilities of the underlying programming languages. A consensus and commitment with respect to language are crucial for developing ESMF in the long run.
- A practical solution must *balance* cultural and technical factors. It must balance the high value of legacy Fortran code and Fortran expertise with the flexibility and extensibility of OO languages and their widespread use in industry.
- A practical solution must *blend* OO and functional design approaches: OO design and flexibility for the framework and functional design and performance for the physics.
- An ideal framework would blend the underlying OO architecture of ROOT and/or Overture, the developmental and language scope of Cactus, the optimization approach of Overture, the coupling design of DDB, the performance of Fortran 90, the documentation and configuration management of Flux Coupler, and the utilities of GEMS and FMS.
- If Cactus *did* have climate-ready features immediately available it is almost certain that the climate community would want to use it, even if it meant learning something new (Cactus system and control words) and making modifications to the upper layers of their model software in Fortran. However they must be willing to learn something new and make *some* changes to their software. If the climate community is unwilling or unable to migrate to an OO language for the framework, then Cactus may be a good choice.

3.4 References

[1] Zebiak, Stephen and Robert Dickinson, 1998: Report of the NSF/NCEP Workshop on Global Weather and Climate Modeling, Executive Summary, *NSF/NCEP Workshop on Global Weather and Climate Modeling*, Aug 5-6. URL: http://nsipp.gsfc.nasa.gov/infra/report.final.html

[2] Common Modeling Infrastructure Working Group (CMIWG). URL: http://janus.gsfc.nasa.gov/~mkistler/infra/master.html

[3] Staff, NASA, 2000: NASA HPCC/ESS Cooperative Agreement Notice (CAN) for Solicitation of Round-3 Grand Challenge Investigations: Increasing Interoperability and Performance of Grand Challenge Applications in the Earth, Space, Life, and Microgravity Sciences.
URL: http://earth.nasa.gov/nra/current/can00oes01/

[4] Fayad, Mohamed E., Douglas C. Schmidt and Ralph E. Johnson, 1999: Building Application Frameworks. *Wiley*.
URL: http://www.amazon.com/exec/obidos/ASIN/0471248754/qid%3D968778251/102-0715276-9734544

[5] Distributed Data Broker (DDB). URL: http://www.atmos.ucla.edu/~drummond/DDB

[6] Flux Coupler Version 4.0 (FC 4.0). URL: http://www.cgd.ucar.edu/csm/models/cpl

[7] Drummond, Tony, 1998: A Partial List of Available Coupling Software, *Infrastructure Working Group Meeting, Tucson, AZ*, Oct 15-16. URL: http://janus.gsfc.nasa.gov/~mkistler/infra/docdir/csw.html

[8] Common Object Request Broker Architecture (CORBA). URL: http://industry.ebi.ac.uk/~corba/

[9] Cactus URL: http://www.cactuscode.org

[10] ROOT URL: http://root.cern.ch/Welcome.html

[11] Benger, Werner, 1999: Web Cactus, *NCSA Cactus Workshop*, Sept 27.
URL: http://www.cactuscode.org/Workshops/NCSA99/talk19/index.htm

[12] Parallel Object-Oriented Methods and Applications (POOMA). URL: http://www.acl.lanl.gov/pooma/

[13] OVERTURE (OVERTURE). URL: http://www.llnl.gov/casc/Overture/

[14] Veldhuizen, Todd, 1995: Expression Templates. *C++ Report*, 7, 26-31. URL: http://extreme.indiana.edu/~tveldhui/papers/Expression-Templates/exprtmpl.html

[15] Velduizen, Todd, 1997: Scientific Computing: C++ Versus Fortran.
URL: http://extreme.indiana.edu/~tveldhui/papers/DrDobbs2/drdobbs2.html

[16] Velduizen, Todd L. and M. Ed Jernigan, 1997: Will C++ Be Faster than Fortran?, *ISCOPE'97*, Aug.
URL: http://www.acl.lanl.gov/iscope97/agenda.html

[17] Parallel Application Work Space (PAWS). URL: http://www.acl.lanl.gov/paws/

[18] A++/P++ (APPPPP).
URL: http://www.llnl.gov/CASC/Overture/henshaw/documentation/App/manual/manual.html

[19] SAGE++ (SAGE++). URL: http://www.extreme.indian.edu/sage/sagexx_ug/sagexx_ug_toc.html

[20] Edison Design Group (EDG). URL: http://www.edg.com

[21] Flexible Modeling System (FMS). URL: http://www.gfdl.gov/

[22] Simplified Wrapper and Interface Generator (SWIG). URL: http://www.swig.org/

[23] Program Database Toolkit (PDT). URL: http://www.cs.uoregon.edu/research/paracomp/pdtoolkit/

[24] Scripting Interface Languages for Object-Oriented Numerics (SILOON).
URL: http://www.acl.lanl.gov/siloon

[25] PERL (PERL). URL: http://www.perl.org

[26] PYTHON (PYTHON). URL: http://www.python.org

ACRONYMS

ACL–Advanced Computing Laboratory
ACM–Atmospheric Chemical Model
AEI–Albert Einstein Institute
AGCM–Atmospheric General Circulation Model
ALICE–Advanced Large-Scale Integrated Computational Environment
ANL–Argonne National Laboratory
ASC–Astrophysics Simulation Collaboratory
CAN–Cooperative Agreement Notice
CCAT–Common Component Architecture Toolkit
CERFACS–European Centre for Research and Advanced Training in Scientific Computation
CERN–European Organization for Nuclear Research
CFD–Computational Fluid Dynamics
CINT–C Interpreter
CL–Communication Library
CMIWG–Common Modeling Infrastructure Working Group
CORBA–Common Object Request Broker Architecture
CSM–Climate System Model
CSU FC–CSU Flux Coupler
CTC–Cornell Theory Center
CVS–Concurrent Version Systems
DAO–Data Assimilation Office
DDB–Distributed Data Broker
DLR–Deutsche Luft- und Raumfahrtzentrum
DOE–Department of Energy
DTL–Data Translation Library
EDG–Edison Design Group
EGRID–European Grid Project
ESM–Earth System Model
ESMF–Earth System Modeling Framework
ESS–NASA Earth and Space Sciences
FMS–Flexible Modeling System
GEMS–Goddard Earth Modeling System
GrACE–Grid Adaptive Computational Engine
GrADS–Grid Application Development Software Project
HPCC–NASA High Performance Computing and Communications
HPF–High Performance Fortran
INRG–International Numerical Relativity Group
I/O–input/output
LANL–Los Alamos National Laboratory
LBL–Lawrence Berkley Laboratories
LLNL–Lawrence Livermore National Laboratory
MCL–Model Communication Library
MPI–Message Passing Interface
MSMRM–Multi-Scale Multi-Resolution Modeling
MSS–Mass Storage System
NCAR–National Center for Atmospheric Research
NCSA–National Center for Supercomputing Applications
NERSC–National Energy Research Scientific Computing Center
NHSE–National HPCC Software Exchange

NOAA–National Oceanic and Atmospheric Administration
NPACI–National Partnership for Advanced Computational Infrastructure
NSF–National Science Foundation
NSIPP–NASA Seasonal to Inter-annual Prediction Project
OGCM–Oceanic General Circulation Model
PAPI–Performance Data Standard and API
PAWS–Parallel Application Workspace
PDE–Partial Differential Equations
PDT–Program Database Toolkit
PETSC–Portable Extensible Toolkit for Scientific Computing
POC–Point of Contact
POOMA–Parallel Object-Oriented Methods and Applications
RB–Registration Broker
RZG–Rechenzentrum Garching der Max-Planck-Gesellschaft
SHMEM–Shared Memory
SILOON–Scripting Interface Languages for Object-Oriented Numerics
STL–Standard Template Library
SUMAA–Scalable Unstructured Mesh Algorithms and Applications
SWIG–Simplified Wrapper and Interface Generator
TAU–Tuning and Analysis Utilities
TIKSL–German Gigabit Testbed Project
UCLA–University of California, Los Angeles
UCSD–University of California, San Diego
UIB–Universitat de les Illes Balears
WUGRAV–Washington University Gravity group
ZIB–Konrad-Zuse-Zentrum fur Informationstechnik Berlin

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE May 2002	3. REPORT TYPE AND DATES COVERED Technical Memorandum

4. TITLE AND SUBTITLE Earth System Modeling Software Framework Survey	5. FUNDING NUMBERS 931
6. AUTHOR(S) Bryan Talbot, Shujia Zhou, and Glenn Higgins	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES) Goddard Space Flight Center Greenbelt, Maryland 20771	8. PEFORMING ORGANIZATION REPORT NUMBER 2001-03976-0
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS (ES) National Aeronautics and Space Administration Washington, DC 20546-0001	10. SPONSORING / MONITORING AGENCY REPORT NUMBER TM—2001–209992

11. SUPPLEMENTARY NOTES

B. Talbot, S. Zhou and G. Higgins: Northrup-Grumman Information Technology/TASC, Chantilly, Virginia

12a. DISTRIBUTION / AVAILABILITY STATEMENT Unclassified–Unlimited Subject Category: 61 Report available from the NASA Center for AeroSpace Information, 7121 Standard Drive, Hanover, MD 21076-1320. (301) 621-0390.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

One of the most significant challenges in large-scale climate modeling, as well as in high-performance computing in other scientific fields, is that of effectively integrating many software models from multiple contributors. A software framework facilitates the integration task, both in the development and runtime stages of the simulation. Effective software frameworks reduce the programming burden for the investigators, freeing them to focus more on the science and less on the parallel communication implementation, while maintaining high performance across numerous supercomputer and workstation architectures.

This document surveys numerous software frameworks for potential use in Earth science modeling. Several frameworks are evaluated in depth, including Parallel Object-Oriented Methods and Applications (POOMA), Cactus (from the relativistic physics community), Overture, Goddard Earth Modeling System (GEMS), the National Center for Atmospheric Research Flux Coupler, and UCLA/UCB Distributed Data Broker (DDB). Frameworks evaluated in less detail include ROOT, Parallel Application Workspace (PAWS), and Advanced Large-Scale Integrated Computational Environment (ALICE). A host of other frameworks and related tools are referenced in this context. The frameworks are evaluated individually and also compared with each other.

14. SUBJECT TERMS Simulation architectures, software framework, survey, climate modeling, computer languages, object-oriented applications.	15. NUMBER OF PAGES 74
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL