Component Specification for Parallel Coupling Infrastructure

J. Walter Larson[1,2] and Boyana Norris[1]

[1] Mathematics and Computer Science Division, Argonne National Laboratory,
Argonne, IL 60439, USA
{larson,norris}@mcs.anl.gov
[2] ANU Supercomputer Facility, The Australian National University, Canberra ACT
0200 Australia

Abstract. *Coupled systems* comprise multiple interacting subsystems and are an increasingly common computational science application, most notably as multiscale and multiphysics models. Parallel computing and, in particular, message-passing programming have enabled the development of these models but also present a *parallel coupling problem* (PCP) in the form of intermodel data dependencies. Component-based software engineering has been proposed as one means of conquering software complexity in scientific applications; and given the compound nature of coupled models, it is a natural approach to addressing the PCP. We define a software component specification for solving the PCP, abstracting the elements of the PCP and mapping them onto a set of components from the Common Component Architecture. We discuss a reference implementation based on the Model Coupling Toolkit. We demonstrate how these components might be deployed to solve coupling problems in climate modeling.

1 Introduction

Multiphysics and multiscale models share one salient algorithmic feature: They involve interactions between distinct models for different physical phenomena. Multiphysics systems entail coupling of distinct interacting natural phenomena; a classic example is a coupled climate model, involving interactions between the Earth's atmosphere, ocean, cryosphere, and biosphere. Multiscale systems bridge distinct and interacting spatiotemporal scales. A good example can be found in numerical weather prediction, where models typically solve the atmosphere's primitive equations on multiple nested and interacting spatial domains. These systems are more generally labeled as *coupled systems*, and the set of interactions between their various parts are called *couplings*.

Though the first coupled climate model was created over 30 years ago [1], their proliferation has been dramatic in the past decade as a result of increased computing power.

On a computer platform with a single address space, coupling introduces algorithmic complexity by requiring data transformation such as integrid interpolation or time accumulation through averaging of state variables or integration of interfacial fluxes.

On a distributed-memory platform, however, the lack of a global address space adds further algorithmic complexity. Since *distributed data* are exchanged between the coupled model's constituent subsystems, their description must include a domain decomposition. If domain decompositions differ for data on source and target subsystems, the data traffic between them involves a communication schedule to route the data from source to destination. Furthermore, all data processing associated with data transformation will in principle involve explicit parallelism. The resultant situation is called the *parallel coupling problem* (PCP) [2,3].

Myriad application-specific solutions to the PCP have been developed [4–10]. Some packages address portions of the PCP (e.g., the M×N problem—see [11]). Fewer attempts have been made to devise a flexible and more comprehensive solution [2,12]. We propose here the development of a *parallel coupling infrastructure* toolkit, PCI-Tk, based on the *component-based software engineering* strategy defined by the Common Component Architecture Forum. We present a component specification for coupling, an implementation based on the Model Coupling Toolkit [2,13,14], and climate modeling as an example application.

2 The Parallel Coupling Problem

We begin with an overview of the PCP. For a full discussion readers should consult Refs. [2,3].

2.1 Coupled Systems

A coupled system $\mathcal{M}$ comprises a set of N subsystem models called *constituents* $\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$. Each constituent $\mathcal{C}_i$ solves its equations of evolution for a set of *state variables* $\boldsymbol{\phi}_i$, using a set of *input variables* $\boldsymbol{\alpha}_i$, and producing a set of *output variables* $\boldsymbol{\beta}_i$. Each constituent $\mathcal{C}_i$ has a spatial domain Γ_i; its boundary $\partial\Gamma_i$ is the portion Γ_i exposed to other models for coupling.

The *state* $\mathbf{U}_i$ of $\mathcal{C}_i$ is the Cartesian product of the set of state variables $\boldsymbol{\phi}_i$ and the domain Γ_i, that is, $\mathbf{U}_i \equiv \boldsymbol{\phi}_i \times \Gamma_i$. The state $\mathbf{U}_i$ of $\mathcal{C}_i$ is computed from its current value and a set of *coupling inputs* $\mathbf{V}_i \equiv \boldsymbol{\alpha}_i \times \partial\Gamma_i$ from one or more other constituents in $\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$. *Coupling outputs* $\mathbf{W}_i \equiv \boldsymbol{\beta}_i \times \partial\Gamma_i$ are computed by $\mathcal{C}_i$ for use by one or more other constituents in $\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$. *Coupling* between between constituents $\mathcal{C}_i$ and $\mathcal{C}_j$ occurs if the following conditions hold:

1. Their computational domains overlap, that is, the *coupling overlap domain* $\Omega_{ij} \equiv \Gamma_i \cap \Gamma_j \neq \emptyset$.
2. They coincide in time.
3. Outputs from one constituent serve as inputs to the other, specifically (a) $\mathbf{W}_j \cap \mathbf{V}_i \neq \emptyset$ and/or $\mathbf{V}_j \cap \mathbf{W}_i \neq \emptyset$ or (b) the inputs $\mathbf{V}_i$ ($\mathbf{V}_j$) can be computed from the outputs $\mathbf{W}_j$ ($\mathbf{W}_i$).

In practice, the constituents are numerical models, and both $\mathit{\Gamma}_i$ and $\partial \mathit{\Gamma}_i$ are discretized[3] by the discretization operator $\hat{\mathbf{D}}_i(\cdot)$, resulting in meshes $\hat{\mathbf{D}}_i(\mathit{\Gamma}_i)$ and $\hat{\mathbf{D}}_i(\partial \mathit{\Gamma}_i)$, respectively. Discretization of the domains $\mathit{\Gamma}_i$ leads to definitions of state, input, and output vectors for each constituent $\mathcal{C}_i$: The *state vector* $\hat{\mathbf{U}}_i$ of $\mathcal{C}_i$ is the Cartesian product of the state variables $\boldsymbol{\phi}_i$ and the discretization of $\mathit{\Gamma}_i$; that is, $\hat{\mathbf{U}}_i \equiv \boldsymbol{\phi}_i \times \hat{\mathbf{D}}_i(\mathit{\Gamma}_i)$. The *input* and *output vectors* of $\mathcal{C}_i$ are defined as $\hat{\mathbf{V}}_i \equiv \boldsymbol{\alpha}_i \times \hat{\mathbf{D}}_i(\partial \mathit{\Gamma}_i)$ and $\hat{\mathbf{W}}_i \equiv \boldsymbol{\beta}_i \times \hat{\mathbf{D}}_i(\partial \mathit{\Gamma}_i)$, respectively.

The types of couplings in $\mathcal{M}$ can be classified as diagnostic or prognostic, explicit or implict. Consider coupling between $\mathcal{C}_i$ and $\mathcal{C}_j$ in which $\mathcal{C}_i$ receives among its inputs data from outputs of $\mathcal{C}_j$. *Diagnostic coupling* occurs if the outputs $\mathbf{W}_j$ used as input to $\mathcal{C}_i$ are computed *a posteriori* from the state $\mathbf{U}_j$. *Prognostic coupling* occurs if the outputs $\mathbf{W}_j$ used as input to $\mathcal{C}_i$ are computed as a forecast based on the state $\mathbf{U}_j$. *Explicit coupling* occurs if there is no overlap in space and time between the states $\mathbf{U}_i$ and $\mathbf{U}_j$. *Implicit coupling* occurs if there is overlap in space and time between $\mathbf{U}_i$ and $\mathbf{U}_j$, requiring a simultaneous, self-consistent solution for $\mathbf{U}_i$ and $\mathbf{U}_j$.

Consider explicit coupling in which $\mathcal{C}_i$ receives input from $\mathcal{C}_j$. The input state vector $\hat{\mathbf{V}}_i$ is computed from a *coupling transformation* $T_{ji} : \hat{\mathbf{W}}_j \rightarrow \hat{\mathbf{V}}_i$. The coupling transformation T_{ji} is a composition of two transformations: a *mesh transformation* $G_{ji} : \hat{\mathbf{D}}_j(\mathit{\Omega}_{ij}) \rightarrow \hat{\mathbf{D}}_i(\mathit{\Omega}_{ij})$ and a *field variable transformation* $F_{ji} : \boldsymbol{\beta}_j \rightarrow \boldsymbol{\alpha}_i$. Intergrid interpolation, often cast as a linear transformation, is a simple example of G_{ji}, but G_{ji} can be more general, such as a spectral transformation or a transformation betweeen Eulerian and Lagrangian representations. The variable transformation F_{ji} is application-specific, defined by the natural law relationships between $\boldsymbol{\beta}_j$ and $\boldsymbol{\alpha}_i$. In general, $G_{ji} \circ F_{ji} \neq F_{ji} \circ G_{ji}$; that is, the choice of operation order $G_{ji} \circ F_{ji}$ versus $F_{ji} \circ G_{ji}$ is up to the coupled model developer and is a source of coupled model uncertainty.

A coupled model $\mathcal{M}$ can be represented as a directed graph $\mathcal{G}$ in which the constituents are nodes and their data dependencies are directed edges. The connectivity of $\mathcal{M}$ is expressible in terms of the nonzero elements of the adjacency matrix A of $\mathcal{G}$. For a constituent's associated node, the number of incoming and outgoing edges corresponds to the number of *couplings*. If a node has only incoming (outgoing) edges, it is a sink (source) on $\mathcal{G}$, and this model may in principle be *run off-line*, using (providing) time history output (input) from (to) the rest of the coupled system. In some cases, a node may have two or more incoming edges, which may require *merging* of multiple outputs for use as input data. For a constituent $\mathcal{C}_i$ with incoming edges directed from $\mathcal{C}_j$ and $\mathcal{C}_k$, merging of data will be required to create $\hat{\mathbf{V}}_i$ if the following conditions hold:

1. The constituents $\mathcal{C}_i$, $\mathcal{C}_j$, and $\mathcal{C}_k$ coincide in time.
2. The coupling domains $\mathit{\Omega}_{ij}$ and $\mathit{\Omega}_{ik}$ overlap, resulting in a *merge domain* $\mathit{\Omega}_{ijk} \equiv \mathit{\Omega}_{ij} \cap \mathit{\Omega}_{ik} \neq \emptyset$.
3. Shared variables exist among the fields delivered from $\mathcal{C}_j$ and $\mathcal{C}_k$ to $\mathcal{C}_i$, namely, $(\boldsymbol{\beta}_j \cap \boldsymbol{\alpha}_i) \cap (\boldsymbol{\beta}_k \cap \boldsymbol{\alpha}_i) \neq \emptyset$.

[3] We will use the terms "mesh," "grid," "mesh points," and "grid points" interchangeably with the term "spatial discretization."

The time evolution of the coupled system is marked by *coupling events*, which either can occur predictibly following a *schedule* or can be *threshhold-triggered* based on some condition satisfied by the constituents' states. In some cases, the set of coupling events fall into a repeatable periodic schedule called a *coupling cycle*. For explicit coupling in which $\mathcal{C}_i$ depends on $\mathcal{C}_j$ for input, the time sampling of the output from $\mathcal{C}_j$ can come in the form of instantaneous values of $\hat{\mathbf{W}}_j$ or as a time integral of $\hat{\mathbf{W}}_j$, the latter used in some coupled climate models [6, 15].

2.2 Consequences of Distributed-Memory Parallelism

The discussion of coupling thus far is equally applicable to a single global address space or a distributed-memory parallel system. Developers of parallel coupled models confront numerous parallel programming challenges within the PCP. These challenges fall into two categories. *Coupled model architecture* encompasses the overall layout of the coupled model's resource allocation and execution scheduling of the constituents. *Parallel data processing* is the set of operations necessary to accomplish data interplay between the constituents.

On distributed-memory platforms, the coupled-model developer faces a strategic decision regarding the mapping of the constituents to processors and the scheduling of their execution. Two main strategies exist—serial and parallel composition [16]. In a *serial composition*, all of the processors available are kept in a single pool, and the system's constituents each execute in turn using all of the processors available. In a *parallel composition* the set of available processors is divided into N disjoint groups called *cohorts*, and the constituents execute simultaneously, each on its own cohort. Serial composition has a simple conceptual design but can be a poor choice if the constituents do not have roughly the same parallel scalability; moreover, it restricts the model implementation to a single executable. Parallel composition offers the developer the option of right-sizing the cohorts based on the constituents' scalability; moreover, it enables the coupled model to be implemented as multiple executables. The chief disadvantage is that the concurrently executing constituents may be forced to wait for input data, causing cascading, hard-to-predict, and hard-to-control execution delays, which can complicate the coupled model's load balance. A third strategy, called *hybrid composition,* involves nesting one within the other to one or more levels (e.g., serial within parallel and vice versa). A fourth strategy, called *overlapping composition,* involves dividing the processor pool such that the constituents share some of the processors in their respective cohorts; this approach may be useful with implicit coupling.

In a single global address space, description and transfer of coupling data are straightforward, and exchanges can be as simple as passing arguments through function interfaces. Standards for field data description (i.e., the $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$) and mesh description for the coupling overlap domains Ω_{ij} (i.e., their discretizations $\hat{\mathbf{D}}_i(\Omega_{ij})$) are sufficient. Distributed memory requires additional data description in the form of a *domain decomposition* $\mathbf{P}_i(\cdot)$, which splits the coupling overlap domain mesh $\hat{\mathbf{D}}_i(\Omega_{ij})$ and associated input and output vectors $\hat{\mathbf{V}}_i$ and

$\hat{\mathbf{W}}_i$ into their respective local components across the processes $\{p_1, p_2, \ldots, p_{K_i}\}$, where K_i is the number of processors in the cohort associated with $\mathcal{C}_i$. That is, $\mathbf{P}_i(\hat{\mathbf{V}}_i) = \{\hat{\mathbf{V}}_i^1, \ldots, \hat{\mathbf{V}}_i^{K_i}\}$, $\mathbf{P}_i(\hat{\mathbf{W}}_i) = \{\hat{\mathbf{W}}_i^1, \ldots, \hat{\mathbf{W}}_i^{K_i}\}$, and $\mathbf{P}_i(\hat{\mathbf{D}}_i(\Omega_{ij})) = \{\hat{\mathbf{D}}_i^1(\Omega_{ij}), \ldots, \hat{\mathbf{D}}_i^{K_i}(\Omega_{ij})\}$.

Consider a coupling in which $\mathcal{C}_i$ receives input from $\mathcal{C}_j$. The transformation T_{ji} becomes a distributed-memory parallel operation, which in addition to its grid transformation G_{ji} and field transformation F_{ji} includes a third operation—*data transfer* H_{ji}. The order of composition of F_{ji}, G_{ji}, and H_{ji} is up to the model developer, and again the order of operations will affect the result. The data transfer H_{ji} will have less of an impact on uncertainties in the ordering of F_{ji} and G_{ji}, its main effect appearing in roundoff-level differences caused by reordering of arithmetic operations if computation is interleaved with the execution of H_{ji}. In addition, the model developer has a choice in the *placement* of operations, that is, on which constituent's cohort the variable and mesh transformations should be performed—the source, $\mathcal{C}_j$, the destination, $\mathcal{C}_i$, on a subset of the union of their cohorts, or someplace else (i.e., delegated to another constituent—called a *coupler* [4]—with a separate set of processes).

2.3 PCI Requirements

The abstraction of the PCP described above yields two observations: (1) the architectural aspects of the problem form a large decision space; and (2) the parallel data processing aspects of the problem are highly amenable to a generic software solution. Based on these observations, a parallel component infrastructure (PCI) must be modular, enabling coupled model developers to choose appropriate components for their particular PCP's. The PCI must include decomposition descriptors for each constituent's mesh for its domain boundary $\mathbf{P}_i(\hat{\mathbf{D}}_i(\partial\Gamma_i))$ and inputs $\mathbf{P}_i(\hat{\mathbf{V}}_i)$ and outputs $\mathbf{P}_i(\hat{\mathbf{W}}_i)$. The PCI must provide communications scheduling for parallel data transfers and transposes needed to implement the H_{ij} operations for each model coupling interaction. Data transformation for coupling is an open-ended problem: Support for variable transformations F_{ij} will remain application-specific. The PCI should provide generic infrastructure for spatial mesh transformations G_{ij}, perhaps cast as a parallel linear transformation. Other desirable features of a PCI include spatial integrals for diagnosis of flux conservation under mesh transformations, time integration registers for time averaging of state data and time accumulation of flux data for implementing loose coupling, and a facility to merge output from multiple sources for input to a target constituent.

3 Software Components and the Common Component Architecture

Component-based software engineering (CBSE) [17, 18] is widespread in enterprise computing. A *component* is an atomic unit of software encapsulating some useful functionality, interacting with the outside world through well-defined interfaces often specified in an *interface definition language.* Components are composed into applications, which are executed in a runtime *framework.* CBSE enables software reuse, empowers application developers to switch between subsystem implementations for which multiple components exist, and can dramatically reduce application development time. Examples of commercial CBSE approaches include COM, DCOM, JavaBeans, Rails, and CORBA. Alas, they are not suitable for scientific applications for two reasons: unreasonably high performance cost, especially in massively parallel applications, and inability to describe scientific data adequately (e.g., they do not support complex numbers).

The Common Component Architecture (CCA) [19, 20] is a CBSE approach targeting high-performance scientific computing. CCA's approach is based on explicit descriptions of *software dependencies* (i.e., caller/callee relationships). CCA component interfaces are known as *ports*: *provides* ports are the interfaces implemented, or provided, by a component, while *uses* ports are external interfaces whose methods are called, or used, by a component. Component interactions follow a *peer component model* through a connection between a uses ports and a provides port of the same type to establish a caller/callee relationship (Figure 1 (a)). The CCA specification also defines some special ports (e.g., GoPort, essentially a "start button" for a CCA application). CCA interfaces as well as port and component definitions are described in a SIDL (*scientific interface definition language*) file, which is subsequently processed by a language interoperability tool such as Babel [21] to create the necessary interlanguage glue code. CCA meets performance criteria associated with high-performance computing, and typical latency times for intercomponent calls between components executing on the same parallel machine are on the order of a virtual function call between same-language components; times for interlanguage component interactions are slightly more, but within 1–2 orders of magnitude of typical MPI latency times.

The port connection and mediation of calls is handled by a CCA-compliant framework. Each component implements a `SetServices()` method where the component's uses and provides ports are registered with the framework. At runtime, uses ports are connected to provides ports, and a component can access the methods of a port through a `getPort()` method. A typical port connection diagram for a simple application (which will be discussed in greater detail in Section 6) is shown in Figure 1 (b).

CCA technology has been applied successfully in many application areas including combustion, computational chemistry, and Earth sciences. CCA's language interoperability approach has been leveraged to create multilingual bindings for the Model Coupling Toolkit (MCT) [22], the coupling middleware used by the Community Climate System Model (CCSM), and a Python implementation of the CCSM coupler. The work reported here will eventually be part of the CCA Toolkit.

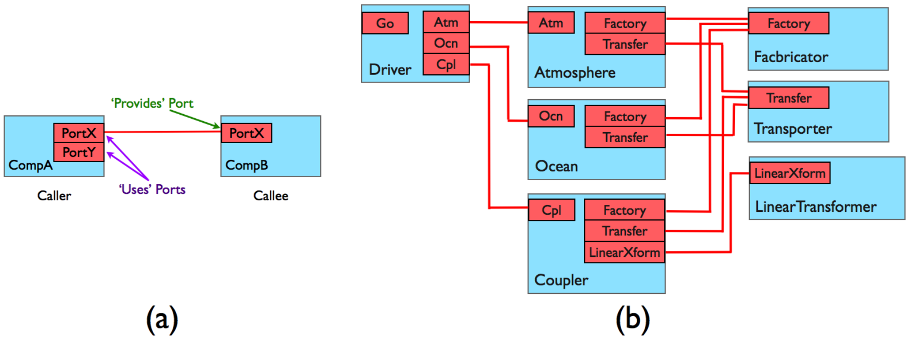

Fig. 1. Sample CCA component wiring diagrams: (a) generic port connection for two components; (b) a simple application composed from multiple components.

4 PCI Component Toolkit

Our PCI specification is designed to address the majority of the requirements stated in Section 2.3. Emphasis is on the parallel data processing part of the PCP, with a middleware layer immediately above MPI that models can invoke to perform parallel coupling operations. A highly modular approach that separates concerns at this low level maximizes flexibility, and this bottom-up design allows support for serial and parallel compositions and multiple executables. We have defined a standard API for distributed data description and constituent processor layout. These standards form a foundation for an API for parallel data transfer and transformation. Below we outline the PCI API and the component and port definitions for PCI-Tk.

4.1 Data Model for Coupling

In our specification, the objects for data description are the SpatialGrid ($\hat{\mathbf{D}}_i(\mathit{\Gamma}_i)$ and $\hat{\mathbf{D}}_i(\partial\mathit{\Gamma}_i)$), the FieldData (the input and output vectors $\hat{\mathbf{V}}_i$ and $\hat{\mathbf{W}}_i$) and the GlobalIndMap (the domain decomposition $\mathbf{P}_i$). This approach assumes a 1-1 mapping between the elements of the spatial discretization and the physical locations in the field data definition. The domain decomposition applies equally to both. For example, a constituent $\mathcal{C}_i$ spread across a cohort of K_i processors, each processor (say, the kth) will have its own SpatialGrid to describe $\hat{\mathbf{D}}_i^k(\partial\mathit{\Gamma}_i)$, and FieldData instantiations to describe its local inputs and outputs $\hat{\mathbf{V}}_i^k$ and $\hat{\mathbf{W}}_i^k$, respectively. In the interest of generality and minimal burden to PCI implementers, we have adopted explicit *virtual linearization* [2, 11, 23–26] as our index and mesh description standard. Virtual linearization supports decomposition description of multidimensional index spaces and meshes, both structured and unstructured. We have adopted an explicit, segmented domain decomposition [2, 11] of the linearized index space.

Data transfer within PCI requires a description of the constituents' cohorts and communications schedules for interconstituent parallel data transfers and intracohort parallel data redistributions. Mapping of constituent processor pools is described by the CohortRegistry interface, which provides MPI processor ID ranks for a constituent's processors within its own MPI communicator and a union communicator of all model cohorts. The CohortRegistry provides lookup services necessary for interconstituent data transfers. Our PCI data model provides two descriptors for the transfer operation H_{ji}: The TransferSched API is an interface that encapsulates interconstituent parallel data transfer scheduling; that is, it contains all of the information necessary to execute all of the MPI point-to-point communication calls needed to implement the transfer.

The data transformation part of the PCI requires data models for linear transformations and for time integration. The LinearTransform encapsulates the whole transformation from storage of transformation coefficients to communications scheduling required to execute the parallel transformation. The TimeIntRegister describes time integration and averaging registers required for loose coupling in which state averages and flux integrals are exchanged periodically for incremental application.

In the SIDL PCI API, all of the elements of the data model are defined as *interfaces*; their implementation as classes or otherwise is at the discretion of the PCI developer.

4.2 PCI-Tk Components

Data Description

The Fabricator Component The Fabricator creates objects used in the interfaces for all the coupling components, along with their associated service methods. It also handles overall MPI communicator management This component has a single provides port, Factory, on which all of the create/destroy, query, and manipulation methods for the coupling data objects reside.

Data Transfer Data under transfer by our PCI interfaces is described by our FieldData specification.

The Transporter Component The Transporter performs one-way parallel data transfers such as the data routing between source and destination constituents, with communications scheduling described by our TransferSched interface. It has one provides port, Transfer, on which methods for both blocking (`PCI_Send()`, `PCI_Recv()`) and nonblocking (`PCI_ISend()`, `PCI_IRecv()`) parallel data transfers are implemented, making it capable of supporting both serial and parallel compositions.

The Transposer Component The Transposer performs two-way parallel data transfers such as data redistribution within a cohort, or two-way data traffic between constituents, with communications scheduling defined by the TransposeSched interface. It has one provides port, Transpose, that implements a data transpose function PCI_Transpose().

Data Transformation The data transformation components in PCI-Tk act on FieldData inputs and, unless otherwise noted, produce outputs described by the FieldData specification.

The LinearTransformer Component The LinearTransformer performs parallel linear transformations using user-defined, precomputed transform coefficients. It has a single provides port, LinearXForm, that implements the transformation method PCI_ApplyLinearTransform().

The TimeIntegrator Component The TimeIntegrator performs temporal integration and averaging of FieldData for a given constituent, storing the ongoing result in a form described by the TimeIntRegister specification. It has a single provides port, TimeInt, that implements methods for time averaging and integration, named PCI_TimeIntegral() and PCI_TimeAverage(), respectively. Users can retrieve time integrals in FieldData form from a query method associated with the TimeIntRegister interface.

The SpatialIntegrator Component The SpatialIntegrator performs spatial integrals of FieldData on its resident SpatialGrid. It has a single provides port, SpatialInt, that offers methods PCI_SpatialIntegral() and PCI_SpatialAverage() that perform multifield spatial integrals and averages, respectively. This port also has methods for performing simultaneously paired multifield spatial integrals and averages; here *pairing* means that calculations for two different sets of FieldData on their respective resident SpatialGrid objects are computed. This functionality enables efficient, scalable diagnosis of conservation of fluxes under transformation from source to target grids.

The Merger Component The Merger merges data from multiple sources that have been transformed onto a common, shared SpatialGrid. It has a single provides port, Merge, on which merging methods reside, including PCI_Merge2(), PCI_Merge3(), and PCI_Merge4() for merging of data from two, three, and four sources, respectively. An additional method PCI_MergeIn() supports higher-order and other user-defined merging operations.

5 Reference Implementation

We are using MCT to build a reference implementation of our PCI specification. MCT provides a data model and library support for parallel coupling. Like the specification, MCT uses virtual linearization to describe multidimensional index spaces and grids. MCT's Fortran API is described in SIDL, and Babel has been used to generate multilingual bindings [22], with Python and C++ bindings and

Table 1. Correspondence between PCI Data Interfaces and MCT Classes

Functionality	PCI Interface	MCT Class
Mesh Description $\hat{\mathbf{D}}_i\Gamma_i$, $\hat{\mathbf{D}}_i(\partial\Gamma_i)$	SpatialGrid	GeneralGrid
Field Data $\hat{\mathbf{U}}_i$, $\hat{\mathbf{V}}_i$, $\hat{\mathbf{W}}_i$	FieldData	AttrVect
Domain Decomposition $\mathbf{P}_i$	GlobalIndMap	GlobalSegMap
Constituent PE Layouts	CohortRegistry	MCTWorld
One-Way Parallel Data Transfer Scheduling H_{ij}	TransferSched	Router
Two-Way Parallel Data Transpose Scheduling H_{ij}	TransposeSched	Rearranger
Linear Transformation G_{ij}	LinearTransform	SparseMatrix SparseMatrixPlus
Time Integration Registers	TimeIntRegister	Accumulator

example codes available from the MCT Web site. The data model from our PCI specification maps readily onto MCT's classes (see Table 1).

The port methods are implemented in some cases through direct use (via glue code) of MCT library routines, and at worst via lightweight wrappers that perform minimal work to convert port method arguments into a form usable by MCT. Table 2 shows in broad terms how the port methods are implemented.

Table 2. Correspondence between PCI Ports and MCT Methods

Component / Port	MCT Method
Fabricator / Factory	Create, destroy, query, and manipulation methods for GeneralGrid, AttrVect, GlobalSegMap, MCTWorld, Router, Rearranger, SparseMatrix, SparseMatrixPlus, and Accumulator
Transporter / Transfer	Transfer Routines `MCT_Send()`, `MCT_Recv()`, `MCT_ISend()`, `MCT_IRecv()`,
Transposer / Transpose	`Rearrange()`
LinearTransform / LinearXForm	SparseMatrix-AttrVect Multiply `sMatAvMult()`
SpatialIntegrator / Spatial Integral	`SpatialIntegral()` and `SpatialAverage()`
TimeIntegrator / TimeIntegral	`accumulate()`
Merger / Merge	`Merge()`

6 Deployment Examples

We present three examples from climate modeling in which PCI-Tk components could be used to implement parallel couplings. the field of coupled climate modeling. In each example, the system contains components for physical subsystems and a *coupler*. The coupler handles the data transformation, and the models interact via the coupler purely through data transfers—a *hub-and-spokes* architecture [15].

The MCT toy climate coupling example comprises atmosphere and ocean components that interact via a coupler that performs intergrid interpolation, and computes application-specific variable transformations such as computation of interfacial radiative fluxes. It is a single executable application; the atmosphere, ocean, and coupler are procedures invoked by the MAIN driver application. It is a parallel composition; parallel data transfers between the cohorts are required. A CCA wiring diagram of this application using PCI-Tk components is shown in Figure 1 (b). The driver component with the Go port signifies the single executable, and this component has uses ports labeled Atm, Ocn, and Cpl implemented as provides ports on the atmosphere, ocean, and coupler components, respectively. The PCI-Tk data model elements used in the coupling are created and managed by the Fabricator, via method calls on its Factory port. The parallel data transfer traffic between the physical components and the coupler are implemented by the Transporter component via method calls on its Transfer port. A LinearTransform component is present to implement interpolation between the atmosphere and ocean grids; the coupler performs this task via method calls on its LinearXform port.

The *Parallel Climate Model (PCM) example* [6] shown in Figure 2 (a) is a single executable and a serial composition. A driver coordinates execution of the individual model components. Since the models run as a serial composition, coupling data can be passed across interfaces, and transposes performed as needed; thus there is a Transpose component rather than a Transfer component. The coupler in this example performs the full set of transformations found in PCM: intergrid interpolation with the LinearTransform; diagnosis of flux conservation under interpolation with the SpatialIntegrator; time integration of flux and averaging of state data using the TimeIntegrator; and merging of data from multiple sources with the Merger. The TimeIntegrator is invoked by both the ocean and coupler components because of the loose coupling between the ocean and the rest of PCM; the atmosphere, sea-ice, and land-surface models interact with the couple hourly, but the ocean interacts with the coupler once per model day. The coupler integrates the hourly data from the atmosphere, land, and sea-ice that will be passed to the ocean. The ocean integrates its data from each timestep over the course of the model day for delivery to the coupler.

The *CCSM example* is a parallel composition. Its coupling strategy is similar to that in PCM in terms of the parallel data transformations and implementation of loose coupling to the ocean. CCSM uses a *peer communciation* model, however, with each of the physical components communicating in parallel with the coupler. These differences are shown in Figure 2 (b). The atmosphere, ocean, sea-ice, land-surface, and coupler are separate executables and have Go ports on them; and the parallel data transfers are implemented by the Transfer component rather than the Transpose component.

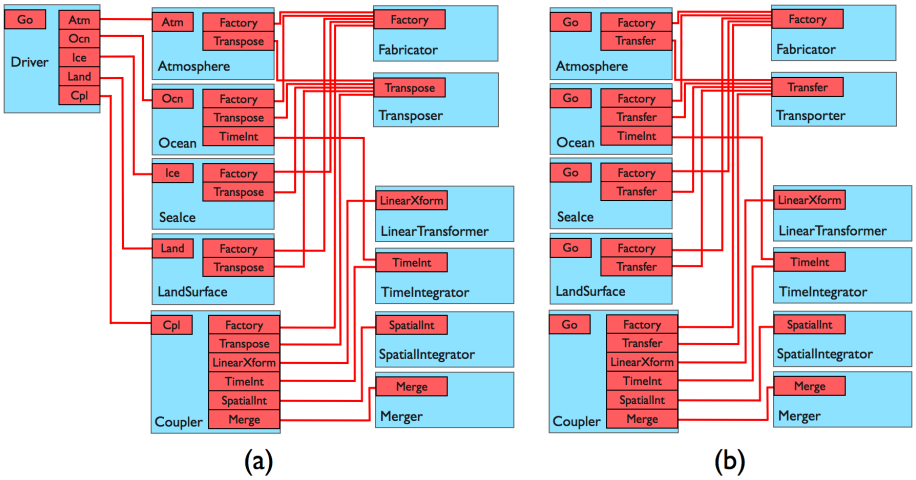

Fig. 2. CCA wiring diagrams for a two coupled climate model architectures: (a) PCM, with serial composition and single executable; (b) CCSM, with parallel composition and multiple executables.

7 Conclusions

Coupling and the PCP are problems of central importance as computational science enters the age of multiphysics and multiscale models. We have described the theoretical underpinnings of the PCP and derived a core set of PCI requirements. From these requirements, we have formulated a PCI component interface specification that is compliant with the CCA, a component approach suitable for high-performance scientific computing—a *parallel coupling infrastructure toolkit* (PCI-Tk). We have begun a reference implementation based on the Model Coupling Toolkit MCT. Use-case scenarios indicate that this approach is highly promising for climate modeling applications, and we believe the reference implmentation will perform approximately as well as MCT does: the component overhead introduced by CCA has been found to be acceptably low in other application studies ([19]); and our own performance studies on our Babel-generated C++ and Python bindings for MCT show minimial performance impact (at most a fraction of a percent versus the native Fortran implementation [22]).

Future work includes completing the reference implementation and a thorough performance study; prototyping of applications using the MCT-based PCI-Tk; modifying the specification if necessary; and exploring alternative PCI component implementations (e.g., using mesh and field data management tools from the DOE-supported Interoperable Technologies for Advanced Petascale Simulations (ITAPS) center [27].

Acknowledgments: This work is primarily a product of the Center for Technology for Advanced Scientific Component Software (TASCS), which is supported by the U.S. Department of Energy (DOE) Office of Advanced Scientific Computing Research through the Scientific Discovery through Advanced Computing Program. Argonne National Laboratory is operated for the DOE by UChicago Argonne, LLC, under Contract No. DE-AC02-06CH11357. The ANU Supercomputer Facility is funded in part by the Australian Department of Education, Science, and Training through the Australian Partnership for Advanced Computing (APAC).

References

1. Manabe, S., Bryan, K.: Climate calculations with a combined ocean-atmosphere model. Journal of the Atmospheric Sciences **26**(4) (1969) 786–789
2. Larson, J., Jacob, R., Ong, E.: The Model Coupling Toolkit: A new Fortran90 toolkit for building multi-physics parallel coupled models. Int. J. High Perf. Comp. App. **19**(3) (2005) 277–292
3. Larson, J.W.: Some organising principles for coupling in multiphysics and multiscale models. Preprint ANL/MCS-P14140207, Mathematics and Computer Science Division, Argonne National Laboratory (2006)
4. Bryan, F.O., Kauffman, B.G., Large, W.G., Gent, P.R.: The NCAR CSM flux coupler. NCAR Tech. Note 424, NCAR, Boulder, CO (1996)
5. Jacob, R., Schafer, C., Foster, I., Tobis, M., Anderson, J.: Computational design and performance of the Fast Ocean Atmosphere Model. In Alexandrov, V.N., Dongarra, J.J., Tan, C.J.K., eds.: Proc. 2001 International Conference on Computational Science. Volume 2073 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2001) 175–184
6. Bettge, T., Craig, A., James, R., Wayland, V., Strand, G.: The DOE Parallel Climate Model (PCM): The Computational Highway and Backroads. In Alexandrov, V.N., Dongarra, J.J., Tan, C.J.K., eds.: Proc. International Conference on Computational Science (ICCS) 2001. Volume 2073 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2001) 148–156
7. Drummond, L.A., Demmel, J., Mechose, C.R., Robinson, H., Sklower, K., Spahr, J.A.: A data broker for distirbuted computing environments. In Alexandrov, V.N., Dongarra, J.J., Tan, C.J.K., eds.: Proc. 2001 International Conference on Computational Science. Volume 2073 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2001) 31–40
8. Valcke, S., Redler, R., Vogelsang, R., Declat, D., Ritzdorf, H., Schoenemeyer, T.: OASIS4 user's guide. PRISM Report Series 3, CERFACS, Toulouse, France (2004)
9. Hill, C., DeLuca, C., Balaji, V., Suarez, M., da Silva, A., the ESMF Joint Specification Team: The architecture of the earth system modeling framework. Computing in Science and Engineering **6** (2004) 18–28
10. Toth, G., Sokolov, I.V., Gombosi, T.I., Chesney, D.R., Clauer, C.R., Zeeuw, D.D., Hansen, K.C., Kane, K.J., Manchester, W.B., Oehmke, R.C., Powell, K.G., Ridley, A.J., Roussev, I.I., Stout, Q.F., Volberg, O., Wolf, R.A., Sazykin, S., Chan, A., Yu, B., Kota, J.: Space weather modeling framework: A new tool for the space science community. Journal of Geophysical Research **110** (2005) A12226
11. Bertrand, F., Bramley, R., Bernholdt, D.E., Kohl, J.A., Sussman, A., Larson, J.W., Damevski, K.B.: Data redistribution and remote method invocation for coupled components. Journal of Parallel and Distributed Computing **66**(7) (2006) 931–946

12. Joppich, W., Kurschner, M., the MpCCI Team: MpCCI - a tool for the simulation of coupled applications. Concurrency and Computation: Practice and Experience **18**(2) (2006) 183–192
13. Jacob, R., Larson, J., Ong, E.: M×N communication and parallel interpolation in ccsm3 using the Model Coupling Tookit. Int. J. High Perf. Comp. App. **19**(3) (2005) 293–308
14. The MCT Development Team: Model Coupling Toolkit (MCT) web site. `http://www.mcs.anl.gov/mct/` (2007)
15. Craig, A.P., Kaufmann, B., Jacob, R., Bettge, T., Larson, J., Ong, E., Ding, C., He, H.: cpl6: The new extensible high-performance parallel coupler for the community climate system model. Int. J. High Perf. Comp. App. **19**(3) (2005) 309–327
16. Foster, I.: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison Wesley, Reading, Massachusetts (1995)
17. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. ACM Press, New York (1999)
18. Heineman, G.T., Council, W.T.: Component Based Software Engineering: Putting the Pieces Together. Addison-Wesley, New York (1999)
19. Bernholdt, D.E., Allan, B.A., Armstrong, R., Bertrand, F., Chiu, K., Dahlgren, T.L., Damevski, K., Elwasif, W.R., Epperly, T.G.W., Govindaraju, M., Katz, D.S., Kohl, J.A., Krishnan, M., Kumfert, G., Larson, J.W., Lefantzi, S., Lewis, M.J., Malony, A.D., Mclnnes, L.C., Nieplocha, J., Norris, B., Parker, S.G., Ray, J., Shende, S., Windus, T.L., Zhou, S.: A component architecture for high-performance scientific computing. Int. J. High Perf. Comp. App. **20**(2) (2006) 163–202
20. CCA Forum: CCA Forum web site. `http://cca-forum.org/` (2007)
21. Dahlgren, T., Epperly, T., Kumfert, G.: Babel User's Guide. CASC, Lawrence Livermore National Laboratory. version 0.9.0 edn. (January 2004)
22. Ong, E.T., Larson, J.W., Norris, B., Jacob, R.L., Tobis, M., Steder, M.: Multilingual interfaces for parallel coupling in multiphysics and multiscale systems. In Shi, Y., ed.: Proc. 2007 International Conference on Computational Science. Volume 4487 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2007) 924–931
23. Lee, J.Y., Sussman, A.: High performance communication between parallel programs. In: Proceedings of 2005 Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models (HIPS-HPGC 2005), IEEE Computer Society Press (April 2005) Appears with the Proceedings of IPDPS 2005.
24. Sussman, A.: Building complex coupled physical simulations on the grid with InterComm. Engineering with Computers **22**(3–4) (2006) 311–323
25. Jones, P.W.: A user's guide for SCRIP: A spherical coordinate remapping and interpolation package. Technical report, Los Alamos National Laboratory, Los Alamos, NM (1998)
26. Jones, P.W.: First and second-order conservative remapping schemes for grids in spherical coordinates. Mon. Wea. Rev. **127** (1999) 2204–2210
27. Interoperable Technologies for Advanced Petascale Simulation Team: ITAPS web site. `http://www.scidac.gov/math/ITAPS.html` (2007)