Chapter 6
The Earth System Modeling Framework

Cecelia DeLuca, Gerhard Theurich and V. Balaji

6.1 Introduction

The Earth System Modeling Framework or ESMF[1] is open source software for building modeling components and coupling them together to form weather prediction, climate, coastal, and other applications. ESMF was motivated by the desire to exchange modeling components amongst centers and to reduce costs and effort by sharing codes. The ESMF package is comprised of a superstructure of coupling tools and component wrappers with standard interfaces, and an infrastructure of utilities for common functions, including calendar management, message logging, grid transformations, and data communications. Infrastructure utilities, including a tool for generation of interpolation weights, can be used independently from the superstructure, enabling users to choose which parts of the software suit their application. The project is distinguished by its strong emphasis on community governance and distributed development.

ESMF was originally designed for tightly coupled applications in the climate and weather domains. Tight coupling is exemplified by the data exchanges between the ocean and atmosphere components of a climate model: a large volume of data is exchanged frequently, and computational efficiency is a primary concern. Such applications usually run on a single computer with hundreds or thousands of proces-

[1] http://www.earthsystemmodeling.org

C. DeLuca (✉)
NOAA/CIRES, Boulder CO USA
e-mail: cecelia.deluca@noaa.gov

G. Theurich
Science Applications International Corporation, McLean VA USA

V. Balaji
Princeton University, New Jersey, USA
e-mail: balaji@princeton.edu

sors, low-latency communications, and a Unix-based operating system. Almost all components in these domains are written in Fortran, with just a few in C or C++.

As the ESMF customer base has grown to include modelers from other disciplines, such as hydrology and space weather, the framework has evolved to support other forms of coupling. For these modelers, ease of configuration, ease of use, and support for heterogeneous coupling may take precedence over performance. Heterogeneity here refers to programming language (Python, Java, etc. in addition to Fortran and C), function (components for analysis, visualization, etc.), grids and algorithms, and operating systems. In response, the ESMF team has introduced support for the Windows platform, and is exploring approaches to language interoperability through web service interfaces. It has introduced more general data structures and several strategies for looser coupling, where components may be in separate executables, or running on different computers.

In this chapter, we describe how coupling fits within the ESMF architecture, and highlight how specific design strategies satisfy user requirements. We then outline how modelers implement ESMF coupling in their applications. Finally, we review alternative coupling strategies that have evolved to suit new communities using ESMF.

6.2 Architectural Overview

The ESMF architecture is based on the concept of components. At its simplest, a software component is a code that has a well-defined calling interface and a coherent function (e.g. Szyperski 2002). Component-based design is a natural fit for climate modeling, since components are ideally suited for the representation of a system comprised of a set of substantial, distinct and interacting domains, such as atmosphere, land, sea ice and ocean. Further, since Earth system domains are often studied and modeled as collections of sub-processes (radiation and chemistry in an atmosphere, for example) it is convenient to model climate applications as an hierarchy of nested components.

Component-based software is also well-suited for the manner in which climate models are developed and used. The multiple domains and processes in a model are usually developed as separate codes by specialists. The creation of a viable climate application requires the integration, testing and tuning of the pieces, a scientifically and technically formidable task. When each piece is represented as a component with a standard interface and behavior, that integration, at least at the technical level, is more straightforward. Similarly, standard interfaces help to foster interoperability of components, and the use of components in different contexts. This is a primary concern for modelers, since they are motivated to explore and maintain alternative versions of algorithms (such as different implementations of the governing fluid equations of the atmosphere), whole physical domains (such as oceans), parameterizations (such as convection schemes), and configurations (such as standalone versions of physical domains).

There are two types of components in ESMF, Gridded Components and Coupler Components. Gridded Components (`ESMF_GridComps`) represent the scientific and computational functions in a model, and Coupler Components (`ESMF_Cpl Comps`) contain the operations necessary to transform and transfer data between them.

Each major physical domain in an ESMF climate or weather model is represented as an ESMF Gridded Component with a standardized calling interface and arguments. Physical processes or computational elements, such as radiative processes or I/O, may also be represented as Gridded Components. ESMF Components can be nested, so that parent components can contain child components with progressively more specialized processes or refined grids.

As a climate or weather model steps forward in time, the physical domains represented by Gridded Components must periodically transfer interfacial fluxes. The operations necessary to couple Gridded Components together may involve data redistribution, spectral or grid transformations, time averaging, and/or unit conversions. In ESMF, Coupler Components encapsulate these interactions.

Design goals for ESMF applications include the ability to use the same Gridded Component in multiple contexts, to swap different implementations of a Gridded Component into an application, and to assemble and extend coupled systems easily; in short, software reuse and interoperability.

A design pattern that addresses these goals is the mediator, in which one object encapsulates how a set of other objects interact (Gamma et al 1995). The mediator serves as an intermediary, and keeps objects from referring to each other explicitly. ESMF Coupler Components follow this pattern. It is an important aspect of the ESMF technical strategy, because it enables the Gridded Components in an application to be deployed in multiple contexts; that is, used in different coupled configurations without changes to their source code. For example, the same atmosphere might in one case be coupled to an ocean in a hurricane prediction model, and in another coupled to a data assimilation system for numerical weather prediction.

Another advantage of the mediator pattern is that it promotes a simplified view of inter-component interactions. The mediator encapsulates all the complexities of data transformation between components. However, this can lead to excessive complexity within the mediator itself, a recognized issue [ibid]. ESMF has addressed this issue by encouraging users to create multiple, simpler Coupler Components and embed them in a predictable fashion in a hierarchical architecture, instead of relying on a single central coupler. This systematic approach is useful for modeling complex, interdependent Earth system processes, since the interpretation of results in a many-component application may rely on a scientist's ability to grasp the flow of interactions system-wide.

Computational environment and throughput requirements motivate a different set of design strategies. ESMF component wrappers must not impose significant overhead, and must operate efficiently on a wide range of computer architectures, including desktop computers and petascale supercomputers. To satisfy these requirements, the ESMF software relies on memory-efficient and highly scalable algorithms (e.g., Devine et al. 2002). Currently ESMF has proven to run efficiently on tens of thousands of processors.

How the components in a modeling application are mapped to computing resources can have a significant impact on performance. Strategies vary for different computer architectures, and ESMF is flexible enough to support multiple approaches. ESMF components can run sequentially (one following the other, on the same computing resources), concurrently (at the same time, on different computing resources), or in combinations of these execution modes. Most ESMF applications run as a single executable, meaning that all components are combined into one program. Starting at a top-level driver, each level of an ESMF application controls the partitioning of its resources and the sequencing of the components of the next lower level.

6.3 Components in ESMF

Both Gridded and Coupler Components are implemented in the Fortran interface as derived types with associated modules. Coupler Components share the same standard interfaces and arguments as Gridded Components. The key data structure in these interfaces is the `ESMF_State` object, which holds the data to be transferred between components. Each Gridded Component is associated with an import State, containing the data required for it to run, and an export State, containing the data it produces. Coupler Components arrange and execute the transfer of data from the export States of producer Gridded Components into the import States of consumer Gridded Components. The same Gridded Component can be a producer or consumer at different times during model execution.

Modelers most frequently write their own Coupler Component internals using ESMF classes bundled with the framework. These classes include methods for time advancement, data redistribution, calculation of interpolation weights, application of interpolation weights via a sparse matrix multiply, and other common functions. ESMF does not currently offer tools for unit transformations or time averaging operations, so users must manage these operations themselves.

Coupler Components can be written to transform data between a pair of Gridded Components, or a single Coupler Component can couple more than two Gridded Components. Multiple couplers may be included in a single modeling application. This is a natural strategy when the application is structured as an hierarchy of components. Each level in the hierarchy usually has its own set of Coupler Components.

The need for modelers to write their own Gridded or Coupler Components has been changing recently. Generic code is being bundled with ESMF that enables reuse of simple Coupler Components and inheritance from prefabricated Gridded Components. A compliance checker has also been introduced that provides feedback on conformance to conventions during the development process. These additions are making compliance easier and improving interoperability among Components.

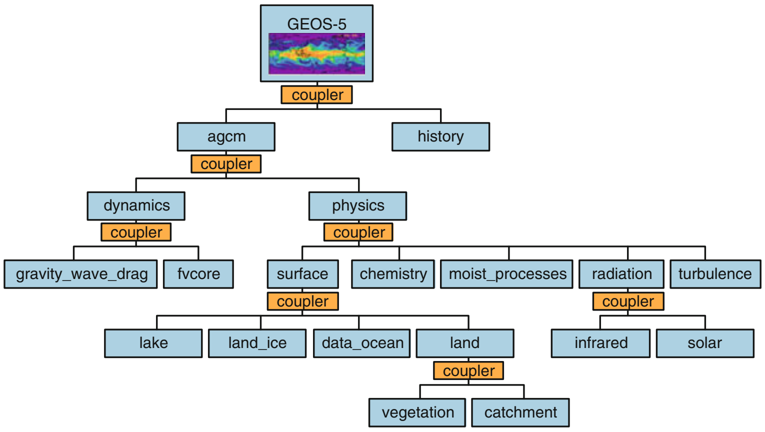

Fig. 6.1 ESMF enables applications such as the GEOS-5 atmospheric general circulation model to be structured hierarchically, and reconfigured and extended easily. Each *box* in this diagram is an ESMF Component

Figure 6.1 shows a simplified schematic of the Goddard Earth Observing System Model, Version 5 (GEOS-5) atmospheric general circulation model, which was constructed in a hierarchical fashion using ESMF. Each box is an ESMF Component. Note that each level in the hierarchy addresses increasingly specific sub-processes represented as Gridded Components, and that each level has its own Coupler Component.

6.4 Remapping in ESMF

Remapping and interpolation in ESMF is accurate, flexible, and fast. ESMF supports a wide variety of grids and remapping options. Generation of interpolation weights and their application is fully parallel. ESMF supports first order conservative, bilinear, and a higher-order finite element-based patch recovery method for remapping in 2D and in some cases 3D. Logically rectangular and unstructured grids are both supported. There is a range of options with respect to masking and the handling of poles and unmapped points. The remapping system is modular; the calculation of interpolation weights can be performed either during a model run or offline, and the application of weights can be made as a separate call.

6.5 Adopting ESMF

It is not necessary to rewrite the internals of model codes to implement coupling using ESMF. Model code attaches to ESMF standard component interfaces via a user-written translation layer that connects native data structures to ESMF data structures. The steps in adopting ESMF are summarized by the acronym *PARSE*:

1. *Prepare user code.* Split user code into initialize, run, and finalize methods and decide on components, coupling fields, and control flow.
2. *Adapt data structures.* Wrap native model data structures in ESMF data structures to conform to ESMF interfaces.
3. *Register user methods.* Attach user code initialize, run, and finalize methods to ESMF Components through registration calls.
4. *Schedule, synchronize, and send data between components.* Write couplers using ESMF redistribution, sparse matrix multiply, regridding, and/or user-specified transformations.
5. *Execute the application.* Run components using an ESMF driver.

In the next two sections, we expand on these steps. The first three steps (*PAR*) focus on wrapping user code in ESMF Components. The last two (*SE*) concern coupling ESMF Components together.

6.5.1 Wrapping User Code in ESMF Components

The first step, *preparing user code*, involves deciding what elements of the application will become Gridded Components. At this time, many climate and weather modeling groups wrap major physical domains (land, ocean, etc.) as Gridded Components, and expect to wrap atmospheric physics and dynamics as Gridded Components in the future. A few applications, such as the GEOS-5 model at the National Aeronautics and Space Administration (NASA), wrap sub-processes such as radiation as Gridded Components as well. A key consideration is what elements of the model are expected to be exchanged or used in multiple contexts; these elements are good candidates for component interfaces. Once Gridded Components are identified, the user must split each of them cleanly into initialize, run, and finalize sections, each callable as a subroutine. These subroutines can have multiple phases; for example, run part one and run part two.

This step also involves analyzing the data flow between components: what fields need to be transferred, what transformations are required between components, how frequently fields must be transferred, and what the data dependencies are. This analysis should give the user a good idea of what Coupler Components will be required, and what operations they should contain. In general, this first step takes the longest.

6.5.2 Adapting Data Structures

The next step in the *PARSE* sequence is *adapting native data structures* to ESMF. Here native model data is copied by reference or value into an ESMF data type. There are multiple ESMF data types that can be used, ranging from simple `ESMF_Array` objects to ESMF_Field objects that store coordinate information and metadata. Many applications contain multiple physical fields that share the same physical domain. Collections of ESMF arrays and fields can be represented in a compact way using ESMF_ArrayBundle and ESMF_FieldBundle objects.

All data exchanged between components is stored in ESMF import and export State objects. These are simple containers that hold ESMF arrays, arraybundles, fields, and fieldBundles. Once native data structures have been associated with ESMF data types, they must be added by the user to the appropriate State objects. At this point the user code is quite close to the required ESMF interfaces. A remaining task is to wrap native calendar and time information into an ESMF_Clock object. The Clock holds information about start time, stop time, time step, and calendar type, and enables the user to set alarms related to specific events. The modeler may also choose to use the ESMF_Config object to store configuration parameters. Config is a straightforward utility that enables the application to read labels and values from a text file.

The resulting user component methods, for initialize, run, and finalize, have ESMF data structures at the calling interface, and look like this example initialize subroutine:

```
subroutine myOcean_Init(gridComp, importState, &
                          exportState, clock, rc)
   type(ESMF_GridComp)  :: gridComp
   type(ESMF_State)     :: importState
   type(ESMF_State)     :: exportState
   type(ESMF_Clock)     :: clock
   integer, intent(out) :: rc

   ! Wrapping layer in which native arrays are extracted
   ! from model data structures, and referenced or copied
   ! into ESMF Arrays, ArrayBundles, Fields, or
   ! FieldBundles. References to these objects are then
   ! placed into import and export States.
   ! Scientific content of initialize routine goes here.

   rc = ESMF_SUCCESS

end subroutine myOcean_Init
```

6.5.3 Registering User Methods

The third step, *registering user methods*, is relatively simple. In it the user-written part of a Gridded or Coupler Component is associated with an ESMF_GridComp or ESMF_CplComp derived type through a special SetServices routine. This is a routine that the user must write, and declare public.

Inside the SetServices routine the user calls ESMF SetEntryPoint methods that associate the standard initialize/run/finalize ESMF Component methods with the names of their corresponding user code subroutines. For example, a user routine called `myOcean_Init` might be associated with the standard initialize routine for a Gridded Component named `myOcean`. The sequence of calls is outlined below.

First the Gridded Component is created. This happens one level above the Gridded Component code. This level may be a relatively small driver program, or it may be a parent Gridded Component. The highest level of an hierarchical ESMF application can be thought of as the "cap". Templates and examples are provided within ESMF to show how the driver is structured.

The application driver would contain code similar to this:

```
type(ESMF_GridComp) :: oceanComp
oceanComp = ESMF_GridCompCreate(name="myOcean", rc=rc)

call ESMF_GridCompSetServices(gridcomp=oceanComp, &
   userRoutine=mySetServices, rc=rc)
```

Here `mySetServices` is the user given name of the public component routine that is responsible for setting the initialize, run and finalize entry points for `ocean Comp`. If the Fortran subroutine names of the user's initialize, run, and finalize methods were `myOcean_Init`, `myOcean_Run`, and `myOcean_Final`, respectively, the `mySetServices` method would contain the following calls:

```
call ESMF_GridCompSetEntryPoint(gridcomp=oceanComp, &
  method=ESMF_SETINIT, userRoutine=myOcean_Init, rc=rc)

call ESMF_GridCompSetEntryPoint(gridcomp=oceanComp, &
  method=ESMF_SETRUN, userRoutine=myOcean_Run, rc=rc)

call ESMF_GridCompSetEntryPoint(gridcomp=oceanComp, &
  method=ESMF_SETFINAL, userRoutine=myOcean_Final,
  rc=rc)
```

These calls link the two pieces of the component: the Gridded Component derived type provided by the framework and the methods provided by the user. The result is that `myOcean` model can be dispatched by a driver or by a parent component in a generic way. The create and destroy operations for components are not linked to user code; they act only on the component derived type.

Like the `ESMF_GridCompCreate()` and `ESMF_GridCompSetServi ces()` calls, the initialize, run, and finalize methods are invoked from a driver or parent component using standard ESMF-defined Component methods. They would follow the `ESMF_GridCompSetServices()` call shown previously:

```
call ESMF_GridCompInitialize(gridcomp=oceanComp, ...,
 rc=rc)
call ESMF_GridCompRun(gridcomp=oceanComp, ..., rc=rc)
call ESMF_GridCompFinalize(grdicomp=oceanComp, ...,
 rc=rc)
```

The omitted arguments indicated by "..." are the optional `importState`, `exportState`, and `clock` arguments. The State arguments are necessary to import and export data to and from the component. The Clock argument provides a means to synchronize the simulation time between different model components.

6.5.4 Coupling Between ESMF Components

A very simple ESMF coupled application might involve an application driver cap, a parent Gridded Component, two child Gridded Components (e.g. an `oceanComp` and an `atmComp`) that require an inter-component data exchange, and two Coupler Components. The hierarchical structure results in calls cascading so that when, for example, the initialize routine of a parent component is called, it contains and calls the initialize routines of its children, which contain and call the initialize routines of their children, and so on. The result is that a call to a Gridded Component initialize method at the top of the hierarchy initializes all the Components in the hierarchy.

The next step, following the *PARSE* approach, involves *scheduling, synchronizing, and sending* data between Components. A sequence similar to that shown for the Gridded Component `oceanComp` would be followed in order to create and register methods for a second Gridded Component `atmComp` and two Coupler Components, `oceanToAtmCpl` and `atmToOceanCpl`.

Assuming `atmComp` needs the temperature field produced by `oceanComp`, the `oceanToAtmCpl` Coupler Component is responsible for the correct data flow. If both Gridded Components define the temperature field on the same physical grid, but with their own custom distribution, a simple field redistribution can be used. Otherwise, if the physical grids are different, an interpolation is necessary.

The required pre-computations for coupling are typically carried out during the Coupler's initialize phase, storing the complete exchange pattern in an `ESMF_RouteHandle` object:

```
type(ESMF_RouteHandle) :: routehandle
ESMF_FieldRedistStore(srcField=oceanTempField, &
   dstField=atmTempField, routehandle=routehandle,
   rc=rc)
```

Here the `oceanTempField` and `atmTempField` are Fields from the ocean Component's export State and the atmosphere Component's import State, respectively. The actual data exchange between these Field objects takes place during the

Coupler’s run phase. A data redistribution call would look like this:

```
call ESMF_FieldRedist(srcField=oceanTempField, &
  dstField=atmTempField, routehandle=routehandle,
   rc=rc)
```

6.5.5 *Executing the Application*

The last step, *execution*, combines all the pieces into a complete ESMF application. The sequencing is specified in the application driver or parent Component. In the following example, the ocean is run first; the ocean to atmosphere coupler communicates the ocean export State to the atmosphere import State; the atmosphere runs; the atmosphere to ocean coupler communicates the atmosphere export State to the ocean import State. This loop repeats until the stop time of the Clock is reached.

```
do while (.not. ESMF_ClockIsStopTime(clock=clock,
 rc=rc))
   call ESMF_GridCompRun(gridcomp=oceanComp,          &
     importState=oceanImportState,                    &
     exportState=oceanExportState, clock=clock, rc=rc)
  call ESMF_CplCompRun(cplcomp=oceanToAtmCpl,         &
     importState=oceanExportState,                    &
     exportState=atmImportState, clock=clock, rc=rc)
  call ESMF_GridCompRun(gridcomp=atmComp,             &
     importState=atmImportState,                      &
     exportState=atmExportState, clock=clock, rc=rc)
  call ESMF_CplCompRun(cplcomp=atmToOceanCpl,         &
     importState=atmExportState,                      &
     exportState=oceanImportState, clock=clock, rc=rc)
  call ESMF_ClockAdvance(clock=clock, rc=rc)
end do
```

6.6 Alternative Forms of Coupling

Two alternative modes of coupling that have been recently introduced are coupling multiple executables and “direct" coupling. In order to couple multiple executables—wholly separate programs—ESMF has been collaborating with the InterComm project (Lee and Sussman 2004). Through an extension of ESMF Array class methods, users can translate ESMF Arrays into their InterComm equivalent, and perform sends and receives of data to InterComm applications. This work is supporting a space weather application in which an ESMF atmosphere couples to an InterComm ionosphere.

Direct coupling was introduced as a way to initiate a data exchange without needing to first return to a Coupler Component interface. The data exchange is arranged within a Coupler Component, usually at initialization time, but it can be invoked from deep within a Gridded Component. This is useful for many modeling situations, including tightly linked physical processes and asynchronous I/O.

6.7 Conclusions and Perspective

ESMF has successfully built a diverse customer base that includes most of the major climate and weather models in the US. Now in its third funding cycle, ESMF is supported by the National Science Foundation (NFS), the Department of Defense, National Oceanic and Atmospheric Administration (NOAA), and NASA. The development team for ESMF and related projects is maintained at about 12 people. It is based at the NOAA Earth System Research Laboratory and the Cooperative Institute for Research in Environmental Science at the University of Colorado. In addition to adding new features, the team provides dedicated user support, nightly regression testing with a suite that includes thousands of tests and examples, and comprehensive documentation.

Timing results for a variety of codes show that the overhead of using ESMF Components is typically negligible (<3% of runtime), and that key operations scale to tens of thousands of processors. Testing performance and exploring new architectures are ongoing activities. Grid remapping and parallel communications are highly scalable and extensible to many new grid types. The framework is very robust and is supported on more than 24 platforms.

ESMF is continuing to add options and optimizations throughout the code. A recent focus has been adding regridding capabilities, including a first order conservative 3D method and higher order conservative method. Looking to the longer term, the ESMF team is exploring the use of metadata to broker and automate coupling services, and to create self-documenting applications. There is also an effort underway to create an option to register ESMF Components for web service interfaces. This will increase accessibility to coupling services for a broader range of scenarios and communities.

References

Devine K, Boman E, Heaphy R, Hendrickson B, Vaughan C (2002) Zoltan: Data management services for parallel dynamic applications. Comput. Sci. Eng. 4(2):90–97

Gamma E, Helm R, Johnson R, Vlissade J (1995) Design patterns: elements of reusable object-oriented software. Addison Wesley, Bostton

Lee J, Sussman A (2004) Efficient communication between parallel programs with intercomm. Technical report CS-TR-4557 and UMIACS-TR-2004-04, Department of Computer Science and UMIACS, University of Maryland

Szyperski C (2002) Component Software. Addison Wesley, Boston