A Graphical Development Tool for Earth System Model Using Component Description Language

Chao Tan, Sujun Cheng, Zhongzhi Luan, Si Ye, Wenjun Li, Depei Qian
Sino-German Joint Software Institute, Beijing Key Laboratory of Network Technology
Beihang University
Beijing, China
E-mail: tanchao128 @126.com

***Abstract*—Historically, researchers have developed large-scale Earth System Model (ESM) applications under the monolithic software development practices that seriously hamper further innovation in complex, highly integrated simulations. Earth System Modeling Framework (ESMF) which organizes applications as discrete components is built to achieve the loose coupling of model and component reuse. Yet it is a big challenge for earth researchers to develop, maintain and share the component code due to the absence of the system software development training and Integrated Development Environments for ESM. To overcome these disadvantages, in this paper we propose an Earth System Model Component Description Language (ESMCDL) which describes not only the interface of component but also the inner behavior of the interface. At the same time, based on this language a graphical development tool is designed to help researchers encapsulate, release components and build templates which consist of these components. Results show that the tool based on the ESMCDL significantly reduces the time required to develop models.**

Keywords-component description language; Earth System Model(ESM); ESM Framework.

I. INTRODUCTION

With the development of Earth System Science, the scale of the software systems for Earth System Model (ESM) becomes increasingly huge. In addition, under monolithic software-development practices, the structure is also becoming more and more complex and there exist a large number of reusable modules as well as their combinations. For example, Community Earth System Model [1], which has nearly 1 million lines of source code, is a coupled climate model for simulating Earth's climate system and composed of one central coupler component and five separate models that simultaneously simulate the Earth's atmosphere, ocean, land, land-ice, and sea-ice, what's more, each model is composed of a serial of physical processes and calculation processes.

NASA with other research institutions has built standards-based open-source software -- Earth System Modeling Framework (ESMF) [2], which defines a component architecture and aims to reduce the coupling between each module and increase Earth System Modeling software reusability.

However, on one hand, researchers who generally lack the system software development training have to consider about plenty of the essential framework code not related with the business logic; on the other hand, there is no unified platform which helps researchers from different institutions share component code. What's more, at present, no matter from general or professional perspective, there is still no widely adopted Integrated Development Environments (IDE) [3] [4] for ESM. Therefore, developing, maintaining and sharing the component code put extra pressure on researchers, which limit the widespread use of ESMF.

In order to solve these problems, this paper firstly introduces the Earth System Model Component Description Language (ESMCDL). The research on component description language can be traced back to the 1980's. The most representative works includes the OBJ [5] and LIL [6] developed by Gougen. In the 1990's, most efforts were spent on how to enable CDLs to describe component, as well as component sub-system. Main works include CIDER [7], RESOLVE [8], and etc. [9] presents a visual Coupling Description Language, but it only focus on hydrology domain. Our ESMCDL, as a kind of metadata describing language, describes not only the interface of the component defined in the ESMF but also the inner behavior of the interface. Moreover, based on this language we develop a graphical development tool which helps researchers not only encapsulate existing modules to form the general earth system component library through the ESMF and parallel technology, but also analyze and summarize the common combinations of components to form the general template library.

This ESMCDL helps the tool with the following functions to achieve the support of rapid development of software.

1) Interface Reuse: ESMCDL separates component code from the abstract definition layer and the specific implementation layer, like the concept of generality. Based on the identical abstract definition, different researchers can adopt different logic realizations.

2) Code Generating: one *.esmcdl file, the context of which follows the ESMCDL format, can be automatically mapped to FORTRAN source code by the tool we provide.

 ISBN: 978-1-61208-184-7

3) Specification Definition: ESMCDL defines one unified programming standard, based on which researchers will realize the encapsulation and release of all components. It can favor code's maintenance and sharing.

4) Template verification: templates could be verified by validating the links between components at the beginning of the design.

The remainder of the paper is organized as follows. Section II gives a short introduction to the ESMF. The syntax structure of the ESMCDL is presented in Section III. Section IV introduces an ESMCDL-based graphical development tool. In Section V, we demonstrate how the tool assist the researchers in developing ESM components as well as templates with a case study of Parallel Ocean Program (POP) [10] and give the preliminary result of it. Finally, we conclude with Section VI.

II. ESMF OVERVIEW

As illustrated in Figure 1, the ESMF comprises a superstructure and an infrastructure. The role of the superstructure layer is to provide a shell that encompasses user code and a context for interconnecting input and output data streams between components. Classes called gridded components, coupler components, and states are used in the superstructure layer to achieve this objective [11]. The infrastructure layer provides a standard support library that researchers can use to speed up construction of components and ensure consistent, guaranteed component behavior. It contains a set of data classes such as Array, Field, and utility classes (including Time, Config). This paper focuses on description for all superstructure classes and data classes.

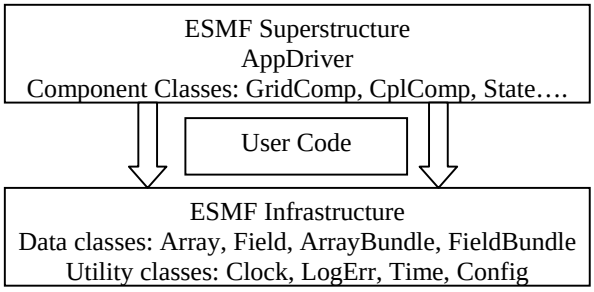

Figure 1. Architecture of the ESMF

Following the principles of loose coupling, we divide the code for each component into Registration, Init, Run, and Final methods as follows:

1) Registration: components register all INF (Init, Run, and Final) methods which can be multi-phase and create all sub-components and States.

2) Init: components allocate the key data structures used to pass persistent states in and out. Coupler components allocate ESMF_RouteHandl Objects.

3) Run: during the run phase, data should be accepted at the beginning of the method and updated after computing in the gridded components. Complex type data transfer occurs by ESMF_RouteHandl Objects in the Coupler components.

4) Final: applications shut down components cleanly in the Final method. For example, Gridded Components destroy sub-components, States and Complex type data, Coupler components destroy ESMF_RouteHandl Object.

Particular emphasis is given to the Sub-component's INF methods, which will be called recursively in each INF method of Gridded Component.

III. THE ESMCDL

In this section we will make rules for the coding behavior of users before introducing the ESMCDL, which simplify the complexity and is also beneficial to the realization of language engine.

1) All the Complex data such as Array, Field, ArrayBundle and FieldBundle should be created in the Init method of which the phase is 1 and destroyed in the Final method.

2) Each State object created in the parent Gridded Component is associated with only one child Gridded Component. It means if one Import State object is passed to any method of one child Gridded Component as parameter, another child Gridded Component couldn't take in it as input.

3) The name of Registration method adopts uniform format "*_setServices", which can be identified by tool.

4) State object (import or export) can only be added into the same type of State object.

5) A Gridded Component contains one Coupler Component at most.

A. Gridded Component

We adopt XML instead of other textual, graphical or binary formats to define ESMCDL as meta–language, since it has many advantages such as scalability, platform-independence and readability. Figure 2 shows the structure of the Gridded Component Description Language.

- **GridComp** denotes an ESMF_GridComp Object. Each GridComp element has *name*, *location* and *variable* property.
- **CplComp** denotes an ESMF_CplComp Object. Each CplComp element has *name*, *location* and *variable* attributes.
- **State** denotes an ESMF_State Object. Parent Gridded Component needs to assign Import State and Export State to its child components. Each State element has a *name*, *variable* and *type* attributes. Type is ESMF_STATE_IMPORT or ESMF_STATE_EXPORT.
- **Init** denotes Gridded Component registers an ESMF_SETINIT type of method in the *_SetSevice subroutine. It consists of four subelements: InitializeComp, StateAdd and AttributeGet, and AttributeSet, whose order of appearance is the same as the order of being called

in the source code. InitializeComp indicates that this Gridded Component will initialize its subcomponents. StateAdd and AttributeSet indicate that the Gridded Component creates data and adds them into state object.

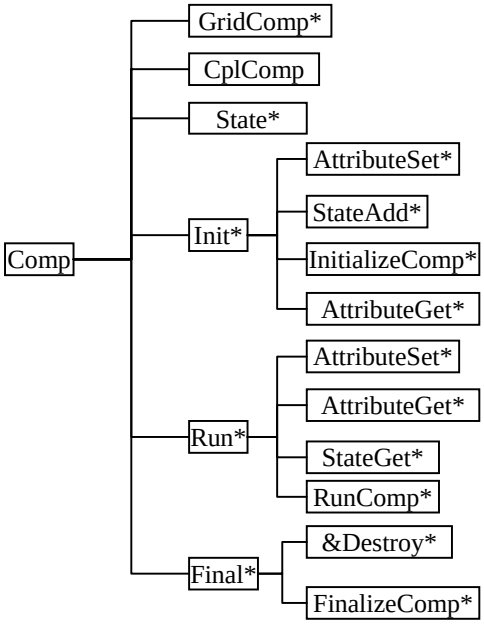

Figure 2. Core Structure of the Gridded Component Description Language

- **Run** denotes Gridded Component registers an ESMF_SETRUN type of method in the *_SetSevice subroutine. It consists of four subelements: RunComp, StateGet, AttributeGet and AttributeSet. RunComp. RunComp indicates that this Gridded Component will run its subcomponent. StateGet and AttributeGet indicate that the Gridded Component needs the data which is necessary to execute.
- **Final** denotes Gridded Component registers an ESMF_SETFINAL type of method in the *_SetSevice subroutine. &∈ {ESMF_GridComp, ESMF_State, ESMF_Array, ESMF_ArrayBundle, ESMF_Field, ESMF_FieldBundle}.

B. Coupler Component

A Coupler Component (or ESMF_CplComp) arranges and executes the data transformation between the Gridded Components. It takes in one or more import ESMF states as input and maps them through spatial and temporal transformation onto one or more output export ESMF states. The structure of the Coupler Component Description Language is given in Figure 3.

- **AttributeCopy** describes the mapping relationship of metadata between the Gridded Components. Scope property (may be one or all) denotes the scope of exchanging. The sub-elements from and to describe where the attributes are from and which to be copied to.
- **&**: There are ESMF_Array, ESMF_Field, ESMF_ArrayBundle, ESMF_FieldBundle on complex data types that have all versions of the data communication methods: Halo, Redist, Regrid, SMMS. Therefore, & can be any of all the 16 transformation methods
- **RouteHandle** denotes an ESMF_RouteHandle Object which identifies those stored communication patterns mentioned above which can be precomputed during an initialization phase and then later executed repeatedly.

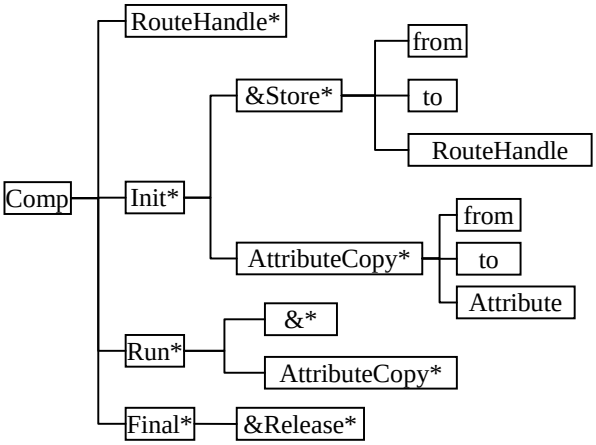

Figure 3. Core Structure of the Coupler Component Description Language

IV. THE GRAPHICAL DEVELOPMENT TOOL

To uniformly build, share components and templates, we created a graphical development tool. This tool is based on ESMCDL which consists of graphical modeling, components publishing and sharing, code generating and data validation. It is built on the Eclipse platform and adopts Photran [12], GMF [13] plug-in technology to strengthen the extensibility. The technical details are beyond the scope of this paper. As shown in Figure 4, the system is composed of four parts.

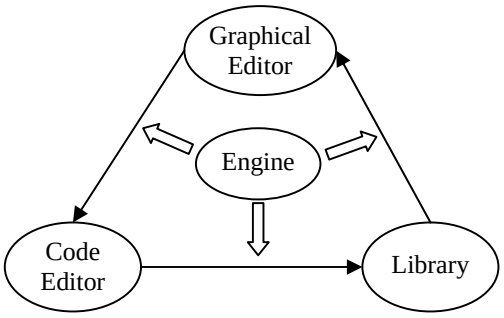

Figure 4. Overview of the graphical tool

 ISBN: 978-1-61208-184-7

1) Graphical editor: application based on the ESMF is executed in a top-down, recursive method mentioned in the section 2. In contrast, with the help of the graphical editor, the application can be built in a bottom-up, iterative method. In the Graphical editor, researchers can do the following things.

*a) Building a Gridded Component (including leaf component and parent component or from a *.esmcdl file directly).*

b) Building a Coupler Component.

c) Building a template: Researchers can drag and drop components which have been in the components library into the graphical editor and then organize them to build the whole template.

2) Code editor: It contains some IDE features such as syntax-highlighting, content assist, code version control and so on.

3) Component and template library: similar to database. It is used to manage ESM templates and components as well as corresponding ESMCDL interface files published by different researchers, including registering, deleting, searching and other functions. The principle that issued once and used many times is followed to provide component providers and consumers with a uniform platform, which make their sharing model code convenient and flexible.

4) ESMCDL engine: ESMCDL needs one corresponding language engine which involves converting ESMCDL documents to Component objects when dragged into the graphical editor or standard FORTRAN code as well as parsing source code of Components to generate ESMCDL files if rule validation is passed.

We present a case study next that ties everything together.

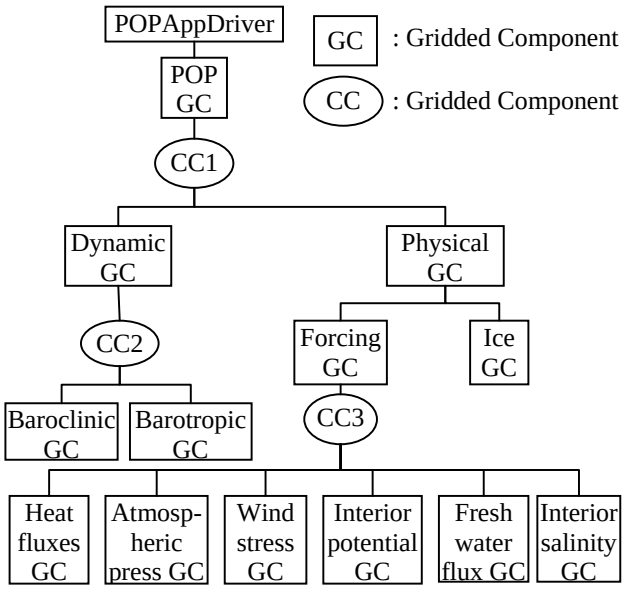

Figure 5. Structure of the components-based POP

V. CASE STUDY

To illustrate the power of our ESMCDL, we now give a complete example, Parallel Ocean Program (POP), one of the most representative models in ESM field. Figure 5 displays the structure of the components-based POP bringing in superstructure and infrastructure for ESMF and following the partitioning strategy mentioned in section 2 as well as programming rules referred in section 3. The Components-based POP is divided into two major parts: Dynamic GC processing data and Physical GC providing date. POPAppDriver is the entry point of the application.

As illustrated in Figure 6 we demonstrate how to create the POP in the bottom-up approach via our graphical software development tool.

First of all, we build those six leaf Gridded Components one by one, including heat fluxes GC, atmospheric pressure GC, wind stress GC, Interior potential GC, fresh water flux GC, interior salinity GC. It includes a series of steps: 1) drag a new component into the **graphical editor** and configure its information such as the name and interface functions. 2) Based on the configuration information, the standard framework code is generated with the help of the **EMCDL engine**. 3) After that, researchers can fill the INF's internal logic code of each child component In the **Code editor**. 4) **EMCDL engine** checks the component code according to the rules to determine whether the code is correct. 5) If no problem, one *.esmcdl file is generated with which every child component will be released to the **component library**. In addition, our tool also supports interface reuse, for instance, based on the same interface different researchers can realize the different calculation processes.

Now those six leaf components have been in our component library. Researchers can drag them into the graphical editor to form the Forcing Gridded Component and then release it into our component library in the same way stated in the previous paragraph. It is necessary to specially emphasize that ESMCDL engine will parse the code of the Component when generating the corresponding *.esmcdl file. We take the forcing Gridded Component and CC1 Coupler Component as an example to explain what the engine analyzes. Figure 7 and 8 illustrate that code is divided into many parts by ESMCDL engine. The code on the left will be mapping to the xml on the right.

Lastly, when all the type components (there may exist different ones based on the same interface) have been prepared in the component library, we can choose some components and drag them into the graphical editor and then link them to build a new POP template or modify an existing POP template saved in template library. Before generating model application code, ESMCDL engine will validate the legitimacy of the link between components recursively from the root node POPAppDriver.

 ISBN: 978-1-61208-184-7

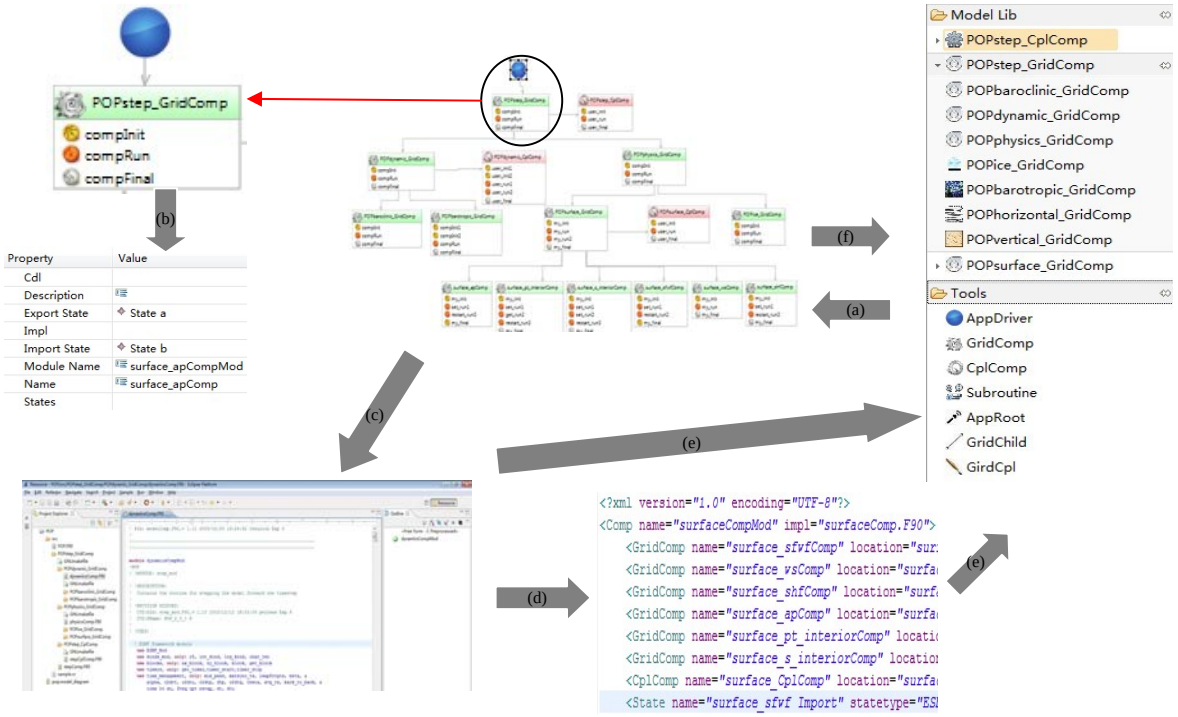

Figure 6. Process of creating one Earth System Model application. (a)Drag and drop components. (b)Configure components' information. (c)Validate the template and generate its code (d)Check the code of the new component and generate its *.esmcdl file. (e)Release the components into the component library (f) Release the template into the templateraray

```
<Comp name="surfaceCompMod"  imp="surfaceComp.F90">
  <GridComp name="sfwf_Comp" location="surface_sfwfCompMod"varaible="sfwf_Comp" />
  ……
  <CplComp name="surfaceCplComp" location="surfaceCplCompMod"variable="surfaceCplComp"/>

  <State name="surface_ws Import" statetype="ESMF_STATE_IMPORT" variable=" wsimp "/>
  <Init name="my_init" phase="1" importState="importState" exportState="exportState">
      <InitializeComp type="Grid "comp="ws_Comp" importState="wsimp" exportState="wsexp" phase="1"/>
      ……
      <StateAdd>
          <state>exportState</state>
          <item type="ESMF_Array" name="FW">FW_array</item>
      </StateAdd>
  </Init>
  <Run name="my_run" phase="1" importState="importState" exportState="exportState">
        <AttributeGet>
          <object>
                exportState
          </object>
          <name>lsmft_avail</name>
          <value type="log_kind">
                lsmft_avail
          </value>
        </AttributeGet>
  <RunComp type="Grid "comp="ws_Comp" importState="wsimp" exportState="wsexp" phase="1"/>
      ……..
  </Run>
  <Final name="my_final" phase="1" importState="importState" exportState="exportState">
      <ArrayDestroy>
          <state>exportState</state>
          <Array name="FW">
            FW_array
          </Array>
      </ArrayDestroy>
      ……
  .</Final>
</Comp>
```

```
//create all child components in register phase
sfwf_Comp=ESMF_GridCompCreate(name="sfwf_name",
grid=parentgrid, rc=rc)

//create all the State objects in register phase
Sfwfimp=ESMF_StateCreate(statetype=
ESMF_STATE_IMPORT, name="surface_sfwf Import")

//add date in to state object in the Init method
call ESMF_StateAdd(exportState, FW_array, rc=rc)

// Register INF method
call ESMF_GridCompSetEntryPoint (gcomp, ESMF_SETRUN,
my  run, rc=rc)

//get date from sfwfexp object
call ESMF_AttributeGet(exportState, name=" lsmft_avail ",
value= lsmft_avail, rc=rc)

//call child component'INF procedures
call ESMF_GridCompRun (ws_Comp, importState=wsimp,
exportState=wsexp, clock=parentclock, rc=rc)

//destroy all date created in Init method
callESMF_StateGet(exportState,"FW",FW_array,rc=rc)
call ESMF_ArrayDestroy(FW_array, rc=rc)
```

Figure 7. Mapping between ESMCDL and code for forcing Gridded Component

 ISBN: 978-1-61208-184-7

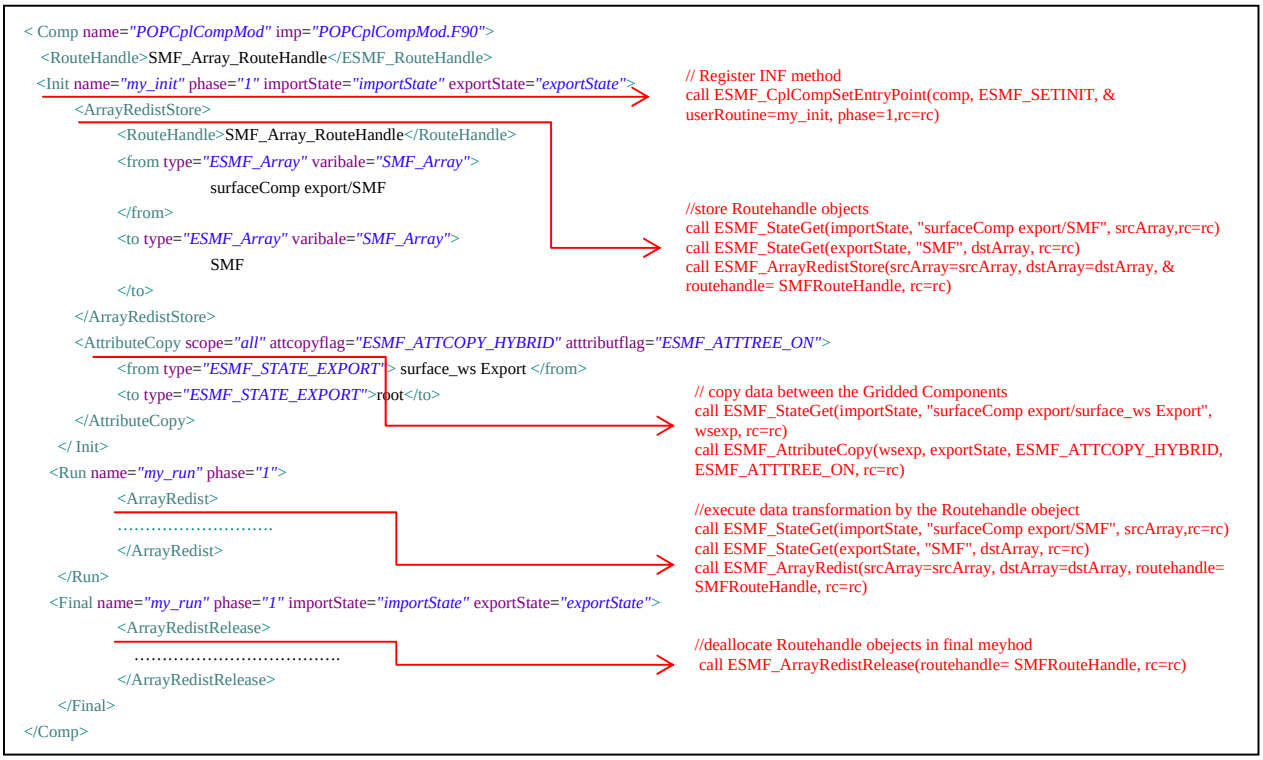

Figure 8. Mapping between ESMCDL and code for CC1 Coupler Componen

We simulated the results of the original POP and the component-based POP. Tests were run on the quad-core Intel(R) Xeon(R) CPU/2.40GHz. The environment of the platform is: RedHat 5.4, MPI: OpenMPI-1.4.1, FORTRAN Compiler: ifort-10.1.022.

We assume that if the K.E. diagnostic and tracer diagnostic in the output log file specified by the name list flag “log _filename” are congruent to both types of POPs; the reconstructed result for the component-based POP is true. The result of the experiments is shown in Table 1.

TABLE I. COMPARISON FOR THE COMPONENT-BASED POP AND ORIGINAL POP AT THE SAME PARALLEL DEGREE.

Parallel Degree	Original POP	Component-based POP	K.E	Tracer
1	7482s	7856s	true	true
2	4042s	4243s	true	true
4	1981s	2083s	true	true

VI. CONCLUSIONS AND FUTURE WORKS

With the development of Earth System Science, Software scale becomes increasingly large. In order to improve the earth system researchers' development efficiency, this paper proposes an Earth System Model Component Description Language and then based on this language introduces a graphical development tool which has been widely used by the earth system researchers from China. We have presented a case study that demonstrates the process of creating one earth system model application by using our graphical development tool. We conclude that the ESMCDL–based graphical development tool not only improves development efficiency for ESM application but also accumulates plenty of reused components and templates which will offer help to other researchers.

In the future, we will continue our research in the following directions: 1) improving the performance of the component-based POP by introducing the parallel technology; 2) expanding the ESMCDL to describe more complex behaviors of a component and logical relationships between components; 3) applying the ESMCDL and the tool to other earth system models to cumulate much more reusable components into our component library.

ACKNOWLEDGMENT

This paper is supported by the 863 project of China under the grant No. 2010AA012404, the China International Science and Technology Cooperation Program from the Ministry of Science and Technology of China under the grant No. 2009DFA12110.

REFERENCES

[1] Kauffman, B.G. and W.G. Large, “The CCSM Coupler Version 5101: Combined User’s Guide, Source Code Reference and Scientific Description,” National Center for Atmospheric Research, 2002, pp. 1-46.

 ISBN: 978-1-61208-184-7

[2] Hill, C., C. DeLuca, V. Balaji, M. Suarez, and A. DaSilva, "The architecture of the earth system modeling framework," Computing in Science and Engineering, 2004, Vol. 6, No.1 pp. 18-28, doi:10.1109/MCISE.2004.1255817.

[3] Eclipse, http://www.eclipse.org, June 2011.

[4] Microsoft Visual Studio, http://www.microsoft.com/visualstudio, July 2011.

[5] Gougen, "Parameterized programming," IEEE Transactions on Software Engineering, 1983, Vol.10, No. 5, pp. 528-543, doi:10.1109/TSE.1984.5010277.

[6] Gougen, "Reusing and interconnecting software component," IEEE Computer, Vol.19, No.2, February 1986, pp. 16-27, doi:10.1109/MC.1986.1663146.

[7] Paolo Bucci and Stephen H. Edwards, "Special Feature: Component-Based Software Using RESOLVE", Software Engineering Notes, ACM SIGSOFT, Vol. 19, No. 4, October 1994, pp.21-67.

[8] Whittle and M. Ratcliffe, "Software Component Interface Description for Reuse," IEEE BCS Software Engineering Journal, Vol.8, No.6. November 1993,pp. 307-318

[9] Tom Bulatewicz and Janice Cuny, "A domain-specific language model coupling," Proceedings of the 2006 Winter Simulation Conference, December 2006. pp. 1091-1100, doi:10.1109/WSC.2006.323199.

[10] Smith R.D. and Gent P., "Reference manual for the Parallel Ocean Program (POP)," Los Alamos Unclassified Report LA-UR-02-2484, 2002.

[11] Earth System Modeling Framework Reference Manual for Fortran Version 5.2, http://www.earthsystemmodeling.org, May 2011

[12] Photran, http://www.eclipse.org/photran/, September 2011.

[13] GMF, http://www.eclipse.org/modeling/gmp/, February 2011.

 ISBN: 978-1-61208-184-7