CLIMATE AND GLOBAL CHANGE

SERIES NO. ANL/CGC-013-0402

Simplifying the Task of Generating Climate Simulations and Visualizations

by

Sheri A. Mickelson,[1] *John A. Taylor,*[1,2] *and Mike Dvorak*[1]

[1]The Computational Institute, University of Chicago, Chicago, IL 60637 and
Mathematics & Computer Science Division,
Argonne National Laboratory, Argonne, IL 60439
[2]Environmental Research Division, Argonne National Laboratory, Argonne, IL 60439

MATHEMATICS AND COMPUTER SCIENCE DIVISION
ARGONNE NATIONAL LABORATORY

Simplifying the Task of Generating Climate Simulations and Visualizations

Sheri A. Mickelson,[1] John A. Taylor,[1, 2] and Mike Dvorak[1]

[1]The Computational Institute, University of Chicago, Chicago, IL 60637 and The Mathematics & Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439
[2]Environmental Research Division, Argonne National Laboratory, Argonne, Illinois 60439
{mickelso, jtaylor, dvorak}@mcs.anl.gov
http://www-climate.mcs.anl.gov

ABSTRACT. TO FULLY EXPLOIT THE USE OF THE MM5 MODELING SYSTEM, THE SCIENTIST MUST SPEND SEVERAL MONTHS STUDYING THE SYSTEM, THUS LOSING VALUABLE RESEARCH TIME. TO SOLVE THIS PROBLEM, WE HAVE CREATED A GRAPHICAL USER INTERFACE, CALLED ESPRESSO, THAT ALLOWS USERS TO CHANGE THESE VALUES WITHOUT HAVING TO EXAMINE THE CODE. THIS APPROACH DRAMATICALLY INCREASES THE USABILITY OF THE MODEL AND SPEEDS PRODUCTIVITY. WE HAVE ALSO MODIFIED VIS5D TO RUN ON TILED DISPLAYS. USING SUCH DISPLAYS ALLOWS US TO VIEW OUR CLIMATE DATA AT MUCH HIGHER RESOLUTION.

1 Introduction

Simulating climate data with the MM5 modeling system [1, 2, 3] is a complex task that is highly error prone and time consuming. In order to complete a simulation, many scripts must be edited and run. Often, the same variable and value must be set in different scripts. If a value is changed erroneously in the script, the run may fail or produce false results. This process wastes precious analysis and computational time. Also a great deal of effort must be devoted to studying the computer code in order to use MM5 to its fullest potential. Typically, it takes many months to learn the model before the scientist can do a simple simulation.

These characteristics of MM5 are found in many model and simulation programs. With the "Gestalt of computers" theory becoming more prevalent every day, people are demanding more user-friendly environments [4]. To this end, we have created a graphical user interface, called Espresso, that makes the task of running climate models more straightforward. With Espresso, the scientist no longer has to be proficient in computer science to perform a simulation, and the likelihood that the model will fail because of errors is greatly reduced. Espresso was also designed with flexibility that allows users to add and delete name list variables as well as load in completely different modeling systems.

Planning Espresso, we considered an object-oriented design to be essential. With this in mind, we chose to write the program in Java. In addition, we incorporated the two packages: Java Swing and the Extensible Markup Language (XML). Swing allows us to add graphical components to the interface quite easily. XML allows users to create new variables easily. An XML configuration file is read in, and Swing components are created to reflect the file.

After a simulation has been created, scientists often wish to visualize the results. To this end, we have enhanced a version of Vis5D that allows scientists to view their data on a high-resolution tiled displays.

2 REQUIREMENTS OF THE ESPRESSO INTERFACE

The Espresso interface was based on an existing interface designed by Veronika Nefedova of the Mathematics and Computer Science Division at Argonne National Laboratory. That interface allowed the users to change a set number of variables, select a task and a machine to run on, and then run the selected task. The interface provided the basic functionality to create a climate simulation, but it needed much more flexibility. The new Espresso Interface can be seen in Figures 1 and 2.

2.1 Flexibility

One of the main goals of Espresso was flexibility. Running atmospheric simulations involves many variables, only some of which may be needed. Therefore, the interface must handle variable selection appropriately.
The Espresso interface was designed to reflect the variables that are in an XML file. As a default XML file is read in by the Java code, variables are stored within a tree structure in memory, and then put into a javax.swing.JTree structure to display graphically. Clicking on the tree brings up the variables on the interface where the values can be modified.
Espresso also provides the option of opening different XML files while the program is running. When a new file is opened, the interface is recreated dynamically. The user can override the values from the default XML file with individual test cases.
As well as opening up a new XML file interface, users can also save the interface as an XML file. This feature allows users to set up a case study, save the variable values, and, if desired, disperse the XML file to other scientists. Results can be duplicated and studied by other researchers easily.

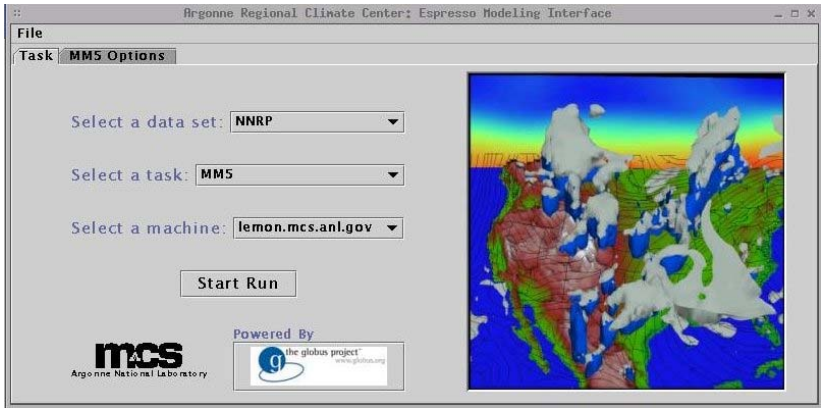

Fig. 1. The first panel of Espresso allows users to select a data set, a task to run, and the machine to run the task on.

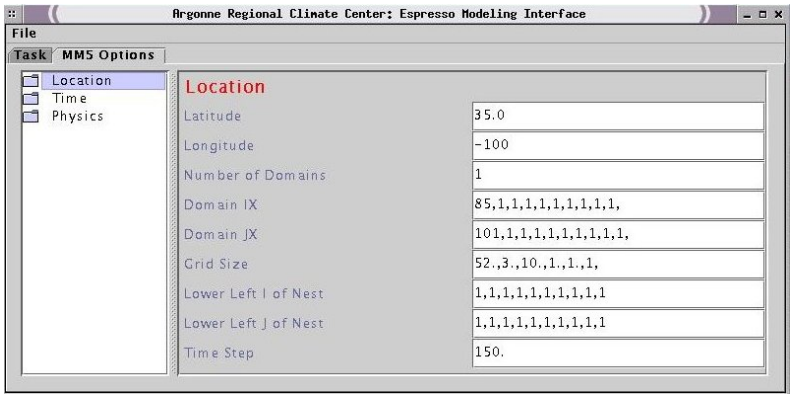

Fig. 2. The second panel displays a tree structure that contains all categories and their variables. When a category is selected, the variables are displayed on the right side where their values can be changed. For compatibility, the format of the data fields are the same as in the original MM5 Fortran namelist.

2.2 Different Modeling Systems

Espresso accommodates different modeling systems. Since the interface parses variables into name lists, it does not matter which model one incorporates into the Espresso modeling system. Some models do require additional information; however, changing the XML file and adding a little Java code can accomplish this task. As we continue to add more modeling systems the need for additional Java code will diminish.

2.3 Object-Oriented Design

An object-oriented design was chosen for easy maintainability. When a program is divided into sections, adding and changing components are simplified. The Espresso interface code was divided into eight classes: a main class, a tabbed pane class, a class for each of the panels, and three classes to implement the parsing and handling of the XML file.

2.4 Error Handling

The previous version of the interface did not check for errors in the input fields. When an incorrect value was entered, the scripts continued to execute, and the user would not be aware of this error until much later. Also, errors were mixed in with a lot of other text output, making them difficult to locate. This situation required significant improvement.

To handle errors in Espresso, we decided to enter parameter ranges and type information in the XML file. For each variable in the XML file, the author must include a minimum and maximum value for the variable and a type (float, integer, array, character, etc). As the XML file is read into the system, these values are checked to make sure they comply with the rules. If a variable does not, a warning dialog box is displayed on the screen and in the terminal output. The same procedure occurs when the user enters invalid input within the GUI, making it much more difficult to enter invalid input into the modeling system.

3 IMPLEMENTATION

While implementing Espresso, we chose the software that was best suited to handle the requirements: Java2 and XML. We used Java2 Swing to create the GUI.

The application first reads in the XML file through the Java tools SAX and DOM. These tools parse the XML file and store the information into a tree data structure. The XML file needed by Espresso provides parameters, values, keywords, minimum and maximum values, and descriptions. The values provided in the XML file are then put through error-handling methods that check whether the values are valid. The graphical interface is then created with Swing from the XML file. The user can interact with this interface as well as change the preset values. After the user selects a data set, task, and machine, the job can be submitted. All the values are then sent to the system's side of Espresso, the scripts are modified, and the jobs are run. The system side of Espresso handles job submission and described in a companion paper, appearing in the volume [5].

3.1.1 Java2

Java was used in the implementation for several reasons. The main reason is its clean object-oriented design and its powerful suite of tools necessary for development of web based applications. To ensure maintainability, we divided Espresso into several different objects. Java also enabled us to use the Swing components to create the interface easily.

Another advantage in using Java is its ability to parse XML files. Specifically, we used Java's SAX and DOM tools. SAX allowed us to read in the file and check for errors that conflicted with the DTD file. DOM allowed us to store the XML information in a tree-linked list in memory. Thus, we were able to traverse the tree quite easily to obtain needed information about the variables.

Espresso is implemented as 100% Java so that we can run across many platforms. This feature increases the flexibility of Espresso as it can be displayed on different types of machine operating systems including Windows and Linux.

3.1.2 Java2 Swing

Swing provides an environment that enables interfaces to be developed quite easily with few commands. Swing includes interface tools such as dialog boxes, text boxes, trees, panels, and tabbed panes that are both consistent and familiar to users [6]. These features increase the ease with which the Espresso user interface can be developed and with which the users can interact.

3.1.3 XML

XML allowed us to create our own format for the configuration file used by the interface. With XML one defines customized tags and how they will be used. Using XML, we defined our own tags to create a readable format. This feature enables the file to be modified easily. It also allows users to add new variables that a user may need to modify on the interface easily. Recompilation and the additional java coding are avoided greatly increasing the flexibility of the Espresso interface. An excerpt of an Espresso XML file is shown in Figure 3.

4 Enhanced Version of Vis5D

Vis5D is a visualization tool created by Bill Hibbard, Johan Kellum, Brian Paul, and others under the Visualization Project at the University of Wisconsin-Madison [7]. This tool visualizes five-dimensional data sets created by climate models. Vis5D provides users with a GUI environment in which to view data objects in a series of different ways [7]. We plan to add the capability to convert model output files from the MM5 and FOAM models to Espresso.

```
<panel>
      <model> MM5 Options
      <category> Location
            <parameter> Latitude
                  <variable type="Float">PHIC</variable>
                  <value>35.0</value>
                  <min>-90.0</min>
                  <max>90.0</max>
                  <description>Central latitude of the coarse
                  domain in degree.</description>
            </parameter>
            <parameter> Longitude
                   <variable type="Float">XLONC</variable>
                   <value>-100</value>
                   <min>-180</min>
                   <max>180</max>
                  <description>Central longitude of the coarse
                  domain in degree.</description>
              </parameter>
            <parameter> Number of Domains
                   <variable type="Integer">MAXNES</variable>
                   <value>1</value>
                   <min>1</min>
                   <max>4</max>
                   <description>Maximum number of domains.</description>
            </parameter>
                  .
                  .
```

Fig. 3. The XML file must specify a model. After the model is specified, parameters are defined and are grouped in categories. Each category will appear in its own pane accessible via the tree.

Although Vis5D provided a convenient way to visualize our data, we needed to visualize our data at a larger scale. Tiled displays provide large viewing areas with increased resolution. Researchers in the Futures Lab in the Mathematics and Computer Science Division at Argonne National Laboratory have developed an Active Mural that consists of fifteen XGA projectors producing one image that contains 4,920 x 2,204 resolution [8]. To use this technology, we modified the Vis5D code and compiled it with the WireGL libraries instead of the OpenGL libraries.

WireGL is a version of OpenGL that replaces the OpenGL commands with its own commands. WireGL divides the image into different sections, which are then dispersed to the correct rendering server [9].

In our early experiments, with help from David Jones in the Mathematics and Computer science Division at Argonne National Laboratory, we determined that not all of the OpenGL calls were accepted by WireGL. Problems occurred with destroying contexts and raster positioning. We found that although many of these calls could simply be removed from the code, this was not the case with the raster positioning problems. In order to place text on the screen, Vis5D makes calls to set a raster position on the screen and then place the text in that location. WireGL is unable to handle this type of call, so inserted a pre-drawn font library into the code.

5 Test Case: The Perfect Storm

In this section we provide an example of how we have used the advanced interfaces that we have been developing at Argonne to perform substantial climate simulations. Currently, Espresso is operational and we have completed some basic test runs in order to demonstrate its capability.

With the previous interface, we created three climate runs of the Perfect Storm: the first with 80 km resolution, the second with 40 km resolution, and the third with 20 km resolution. A snapshot of each resolution is shown in Figure 4.

This interface functioned similarly to Espresso but offered much less flexibility. The interface allowed users to modify the preset values set in a text file and then submitted the values to the MM5 scripts. MM5v3 was run, and the data were visualized with Vis5D.

We hope to produce a 10 km resolution run with a grid size of 400x400 with Espresso. This presents a challenge because the input boundary condition files that are generated can grow larger than 2 gigabytes. To accomplish this task, we must edit MM5 to need input data daily instead of monthly.

6 Conclusions and Future Work

We have created a graphical user interface Espresso that simplifies the task of running climate simulations. Scientists no longer need to dive into the modeling code to change simulations; users can now simply edit an XML file or change values within the interface.

We have used Espresso successfully to create high-resolution climate runs with Espresso. The visualization of these runs presents other software challenges, however. To date, we modified the code of Vis5D to run on tiled displays. Thus, we can see climate details at much higher resolution, aiding in the interpretation of the climate modeling results.

Both applications are in their beginning stages of development. We hope to increase Espresso's flexibility by setting it up for other models such as FOAM and CCM. We also intend to add other modules to the interface. We plan to add a progress screen so users can see the status of the current run. We also need to add an interactive topography file for use with the FOAM model so that users can select which area of the world to simulate, can shift continents around to simulate the Earth's climate millions of years ago, or even simulate of other planets such as Mars.

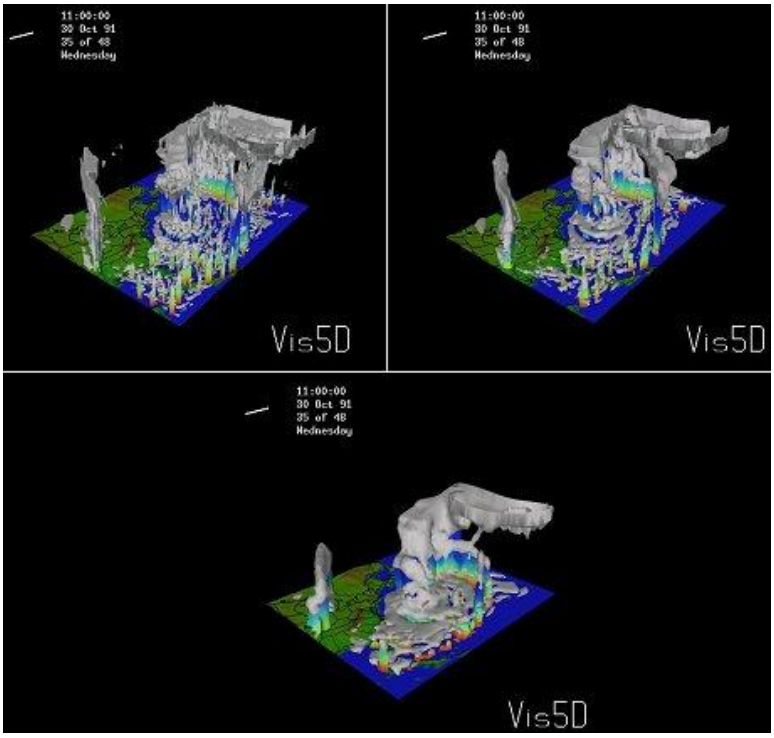

Fig. 4. Clockwise from top left: 20 km resolution climate run with a grid size of 200x250; 40 km resolution climate run with a grid size 100x125; 80 km resolution climate run with a grid size of 50x75. Snapshots are of October 30, 1991, 11:00 AM, of the Perfect Storm climate simulations created with MM5v3. (See http://www-unix.mcs.anl.gov/~mickelso/ for animations of above simulations.)

Vis5D's work involves increasing its functionality. Ultimately we also need to address the problem of the increasing size of the output data sets generated by climate models. New interfaces must be developed to handle this requirement.

Acknowledgments

We thank the staff of the Futures Laboratory at the Mathematics and Computer Science Division at Argonne National Laboratory in Argonne, Illinois. We also thank Veronika Nefedova of the Mathematics and Computer Science Division at Argonne National Laboratory. This work was supported in part by the Office of Biological and Environmental Research, U.S. Department of Energy under Contract W-31-109-ENG-38. This work was also supported by the NSF Information Technology Research Grant, ATM-0121028.

References

1. Chen, F. and J. Dudhia. 2001: Coupling an Advanced Land-Surface/Hydrology Model with the Penn State/NCAR MM5 Modeling system: Part I: Model Implementationand Sensitivity, Monthly Weather Review, in press. (See also Pennsylvania State University/National Center for Atmospheric Research, MM5 Home Page http://www.mmm.ucar.edu/mm5/mm5-home.html
2. Dudhia, J. 1993. A Nonhydrostatic Version of the Penn State/NCAR Mesoscale Model: Validation Tests and Simulation of an Atlantic Cyclone and Cold Front. Mon. Wea. Rev. 121: 1493-1513
3. Grell, G.A., J Dudhia, and D.R. Stauffer. 1994. The Penn State/NCAR Mesoscale Model (MM5). NCAR technical note, NCAR/TN-398_STR, 138 pp.
4. H. Rex Hartson and Deborah Hix. Human-Computer Interface Development: Concepts and Systems for Its Management. ACM Computing Surveys, vol. 21, no. 1. March 1989.
5. Mike Dvorak, John Taylor, Sheri Mickelson (2002). Designing a flexible Grid Enable Scientific modeling Interface. Proc. 2002 International Conference on Computational Science, eds. V.N. Alexandrov, J.J. Dongarra, C. J. K. Tan, Springer-Verlag (in preparation for).
6. Brad Myers, Scott E. Hudson, and Randy Pausch. Past, Present, and Future of User Interface Software Tools. ACM Transactions of Computer-Human Interaction (3-28), vol. 7, no. 1. March 2000.
7. Space Science and Engineering Center University of Wisconsin – Madison, Vis5D Home Page http://www.ssec.wisc.edu/~billh/vis5d.html.
8. Futures Lab, Mathematics & Computer Science, Argonne National Laboratory. ActiveMural http://www-fp.mcs.anl.gov/fl/activemural/.
9. Stanford Computer Graphics Lab, WireGL: Software for Tiled Rendering Home Page http://graphics.stanford.edu/software/wiregl/index.html.