A High-Performance Framework for Sun-to-Earth Space Weather Modeling

Ovsei Volberg
Tech-X Corporation
Boulder, CO 80303
volov@txcorp.com

Gabor Toth
The University of Michigan,
Ann Arbor, MI 48109-2143
gtoth@umich.edu

Tamas I. Gombosi
The University of Michigan
Ann Arbor, MI 48109-2143
tamas@umich.edu

Quentin F. Stout
The University of Michigan,
Ann Arbor, MI 48109-2143
qsout@eecs.umich.edu

Kenneth G. Powell
The University of Michigan
Ann Arbor, MI 48109-2143
powell@umich.edu

Darren De Zeeuw
The University of Michigan,
Ann Arbor, MI 48109-2143
darrens@umich.edu

Aaron J. Ridley
The University of Michigan
Ann Arbor, MI 48109-2143
ridley@umich.edu

Kevin Kane
The University of Michigan,
Ann Arbor, MI 48109-2143
kanekt@umich.edu

Kenneth C. Hansen
The University of Michigan
Ann Arbor, MI 48109-2143
kenhan@umich.edu

David R. Chesney
The University of Michigan
Ann Arbor, MI 48109-2143
chesneyd@eecs.umich.edu

Robert Oehmke
The University of Michigan
Ann Arbor, MI 48109-2143
oehmke@engin.umich.edu

Abstract

The Space Weather Modeling Framework (SWMF) aims at providing software architecture for integrated modeling of different domains of Sun-Earth system and high-performance physics-based space weather simulation. The SWMF component architecture promotes collaboration between developers of individual physics models and empowers them by providing coupling context and parallel and distributed computing support. The framework design places minimal requirements on components. The web-based Graphical User Interface facilitates the remote access to the framework even from scientific groups that do not have an access to supercomputers or clusters otherwise.

1. Introduction

All The Sun-Earth system is a complex natural system of many different interconnecting elements. The solar wind transfers significant mass, momentum and energy to the magnetosphere, ionosphere, and upper atmosphere, and dramatically affects the physical processes in each of these physical domains.

The various domains of the Sun-Earth system can be simulated with stand-alone models if simplifying assumptions are made about the interaction of the particular domain with the rest of the system. These models can be combined into a complex system of mutually interacting models, each associated with one or more physics domains to address a wider range of physical phenomena. For the prediction of extreme space weather events these models must be run and coupled in an efficient manner, so that the simulation can run faster than real-time. The ability to simulate and predict space weather phenomena is important for many applications [1], for instance the success of spacecraft missions and the reliability of satellite communication equipment. In extreme cases, the magnetic storms may have significant effects on the power grids used by millions of households.

Traditionally, researchers developed monolithic space physics applications, which were able to model several domains of the Sun-Earth system, but it was rather difficult to select an arbitrary subset of the various models, to replace one model with another one, to change the coupling schedules of the interacting models, and to run these models concurrently on parallel computers. As an illustrative example of modeling multiple domains of the Sun-Earth system with a monolithic numerical code, we describe the evolution of the space plasma simulation program BATSRUS developed at the University of Michigan. Originally BATSRUS was designed as a very efficient, massively parallel MHD code for space physics applications [2, 3]. It is based on a block adaptive Cartesian grid with block based domain decomposition, and it employs the Message Passing Interface (MPI) standard for parallel execution. Later the code was coupled to an ionosphere model [4] various upper atmosphere models [5] and the inner magnetosphere model [6]. These couplings were done in a highly integrated manner resulting in a monolithic code with the limitations mentioned above. Thus, although BATSRUS is successfully used for the global MHD simulation of space weather [7], monolithic programs like it are not able to support the community effort to share models and to combine these models into flexible, extensible and efficient high-performance applications.

The Center for Space Environment Modeling at the University of Michigan and its collaborators are building a Space Weather Modeling Framework (SWMF) to provide NASA and the modeling community with a high-performance computational tool with "plug-and-play" capabilities to model the physics from the surface of the Sun to the upper atmosphere of the Earth. The SWMF is designed to couple the models of the various physics domains in a flexible yet efficient manner, which makes the prediction of space weather feasible on massively parallel computers. Each model has its own dependent variables, a mathematical model in the form of equations of evolution, and a numerical scheme with an appropriate grid structure and temporal discretization. The physics domains may overlap each other or they can interact with each other through a boundary surface. The SWMF should be able to incorporate models from the community and couple them with minimal changes in the software of an individual model.

The SWMF architecture is based on two contemporary ideas: the framework-based approach combined with the component-based approach [8] to software development (CBSD). Gamma et al [9] define a framework as a set of cooperating classes that make up a reusable design for a specific class of software. The framework captures the design decisions that are common for its application domain: it emphasizes the design reuse over module reuse. The main contribution of the framework is the architecture it defines. From a user's perspective the use of a framework means an inversion of control in comparison to a traditional use of a library or a toolkit: one should reuse the main body and some kernel and write or add the module that the main body calls. The CBSD approach, i.e. the composition of systems from existing or newly developed components, was chosen to address the heterogeneous nature of the computational models of the Sun-Earth system and to satisfy the design goals. The CBSD promises higher quality, lower cost and shorter development time. At the same time the CBSD is a paradigm change in the way the Sun-Earth system is modeled.

In this paper we present the design and implementation of the first working prototype of the SWMF.

2. Architecture overview

The SWMF aims at providing a flexible and extensible software architecture for multi-component physics-based space weather simulations, as well as for various space physics applications [10, 11]. One of the most important features of the SWMF is that it can incorporate different computational physics modules to model different domains of the Sun-Earth system. Each module for a particular domain can be replaced with alternatives, and one can build an application instance with only a subset of the modules if desired.

The first working prototype of the SWMF included components for five physics domains: Global Magnetosphere (GM), Inner Heliosphere (IH), Ionosphere Electrodynamics (IE), Upper Atmosphere (UA) and Inner Magnetosphere (IM):

- The GM and IH components are based on the University of Michigan's BATSRUS MHD module. The highly parallel BATSRUS code uses a 3D block-adaptive Cartesian grid.
- The IM component is the Rice Convection Model (RCM) developed at Rice University. This serial module uses a 2D non-uniform spherical grid.
- The IE component is a 2D spherical electric potential solver developed at the University of Michigan. It can run on 1 or 2 processors since the northern and southern hemispheres can be solved in parallel.
- The UA component is the Global Ionosphere Thermosphere Model (GITM) recently developed at the University of Michigan as a fully parallel 3D spherical model.

Two additional components for new physics domains were added to the SWMF for its second release: the Radiation Belt model developed at Rice University by A. Chan, D. Wolf and Bin Yu and the Solar Energetic Particle model developed at the University of Arizona by J. Kota. With the addition of these two components the SWMF will be able to simulate the space effects of the great importance for radiation protection of spacecraft equipment and for the protection of lives and health of astronauts.

The main SWMF design goals were defined in [12] as (1) incorporate computational physics modules with only modest modification, (2) achieve good parallel performance in the coupling of the physics components, and (3) allow physics components to interact with the SWMF as efficiently as possible.

The efficient coupling of any arbitrary pair of parallel applications, each of them having its own grid structure and data decomposition, is not easily achieved. There are several a priori known problems, which need to be solved so that the heterogeneous computational models of the different domains of the Sun-Earth system can properly inter-operate. An incomplete list of these problems is:

1. There are serial and parallel models.
2. An individual model is usually developed for stand-alone execution.
3. Input/output operations do not take into account potential conflicts with other models.
4. Models often do not have checkpoint and restart capabilities.
5. The majority of models are written in non-object oriented languages (e.g. Fortran 77 and Fortran 90.

We employed the framework-based approach combined with the CBSD approach to develop a multi-purpose, easy-to-use software architecture, which facilitates efficient communications between exchangeable numerical physics models describing Sun-Earth system. The CBSD emphasizes the principle of separation between a component implementation and its interfaces. Major characteristics of CBSD, which are important for the SWMF, include (1) components are the atomic software units of modeling, design and implementation, (2) components interact through interfaces; interfaces are in the center of the architectural design: the loose coupling is essential (3) an interface is separated from the component implementation, and (4) an interface can be implemented by multiple components

There are several potential solutions that provide the necessary interoperability mechanism between parallel modules [13]. The most promising approach is to define a standard set of interface functions that every physics component must provide. In the SWMF a component is created from a physics module, for example BATSRUS, by making some minimal required changes in the module and by adding two relatively small units of code: (1) a wrapper, which provides the standard interface to control the physics module; and (2) a coupling interface, to perform the data exchange with other components.

Each application instance created by the SWMF contains a single Control Module (CON), which controls initialization and execution of the components and is responsible for component registration, processor layout for each component, and coupling schedules.

From a component software technology perspective, both the wrapper and coupling interface are component interfaces: the wrapper is an interface with CON, and the coupling interface is an interface with another component. As shown in Figure 1, the wrapper interface functions have standard names, which makes swapping between various versions of a component possible. Both the wrapper and the coupling interface are constructed from the building blocks provided by the framework. The structure of a component and its interaction with CON and another component are illustrated in Figure 1.

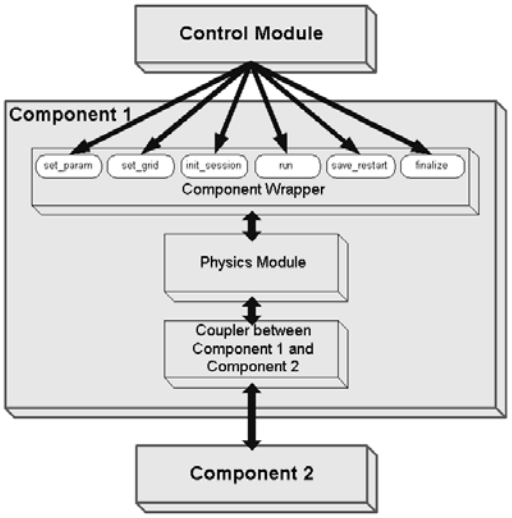


Figure 1. Integration of physics modules into the SWMF architecture

Requirements play a pivotal role in the initial construction of component-based software system. The SWMF is not an exception: each of the SWMF design goals is reflected in the interoperability policy [12]. Specifically, the requirements necessary to ensure the interoperability are formulated as the SWMF compliance definition, which has a dual character: the compliance definition for physics modules and compliance definition for components. The physics modules must comply with a minimum set of requirements before they are transformed into a component [12]:

1. The parallelization mechanism (if any) should employ the MPI standard;
2. The module needs to be compiled as a library that could be linked to an executable;
3. The module should be portable to a specific combination of platforms and compilers, which include Linux workstations and NASA super-computers [12]; The stand-alone module must successfully run a model test suite provided by the model developer on all the required platform/compiler combinations;
4. The module should read input from and write output to files which are in a subdirectory unique for the component;
5. A module should be implemented in Fortran 77 and/or Fortran 90;
6. A module should be supplied with appropriate documentation.

The SWMF requirements for a component are defined in terms of a set of methods to be implemented in the component wrapper shown in Figure 1. The methods enable the component to perform the following tasks:

1. Component registration and parallel setup;
2. Input and verification of the component parameters;
3. Provide grid description to CON;
4. Initialization for session execution;
5. One time step execution (which cannot exceed a specified simulation time);
6. Data exchange with another component via calls to an appropriate coupler;
7. Component state recording into a restart file on request;
8. Finalize the component state at the end of the execution.

The registration provides information about the component to CON, such as the name and the version of the component. In return, CON assigns an MPI communication group to the component based on the processor layout defined in the LAYOUT.in file. An example of this file is shown in Figure 2. The first column identifies the components by their abbreviated names, while the rest of the columns define the first and last processor element (PE) ranks, and the PE stride. In the example shown in Figure 2, the GM component runs on every even processor, the IE component runs on the first two processors, the IH component runs on every odd processor, the IM component runs on processor 11, and the UA component runs on 4 processors from ranks 12 to 15. Only registered components can be used in the run.

```
Name   first last stride
#COMPONENTMAP
GM     0   999     2
IE     0     1     1
IH     1   999     2
IM    11    11     1
UA    12    15     1
#END
```

Figure 2. An example of the LAYOUT.in file

The execution is completed in sessions. In each session the parameters of the framework and the components can be changed. The parameters are read from the PARAM.in file, which may contain further included parameter files. These parameters are read and broadcast by CON and the component specific parameters are sent to the components for reading and checking. The CON related parameters define the initial time, coupling schedules, frequency of saving restart files, final simulation time of the session, and other information, which is not restricted to a single component. At the beginning of each session the components are initialized and the interacting

components are coupled together for the first time. The SWMF application can operate in two different session execution regimes.

The sequential execution regime is backward compatible with BATSRUS. In this regime the components are synchronized at every time step, so typically only one component is executing (possibly on many processors) at any given time. The coupling patterns and schedules are mostly predetermined.

In the concurrent execution regime, the components communicate only when necessary. This is possible because the coupling times are known in advance. The components advance to the coupling time and only the processors involved in the coupling need to communicate with each other. In this execution model all components are 'equal', any physically meaningful subset can be selected for the run, and their coupling schedules are determined by the parameters given in the PARAM.in file. The possibility of deadlocks is carefully avoided.

Based on parameters specified in the PARAM.in file, CON may instructs the components to save their current state into restart files periodically. This makes possible the seamless continuation of a run from a given point of the simulation. Checkpoint restart is an essential feature of a robust, user-friendly, and fault-tolerant software design. At the end of the last session each component finalizes its state. This involves writing out final plot files, closing log files, and printing performance and error reports. After the components have finalized, CON also finalizes and stops the execution.

The framework building blocks are implemented on the base of emulation of Object-Oriented Programming concepts in Fortran 90 [14], mostly as singleton classes [9], i.e. each of them has exactly one instance and provides a global point of access to it.

The coupling of the components is realized either with plain MPI calls, which are specifically designed for each pair of interacting components, or via the general SWMF coupling toolkit. The toolkit can couple components based on the following types of distributed grids:

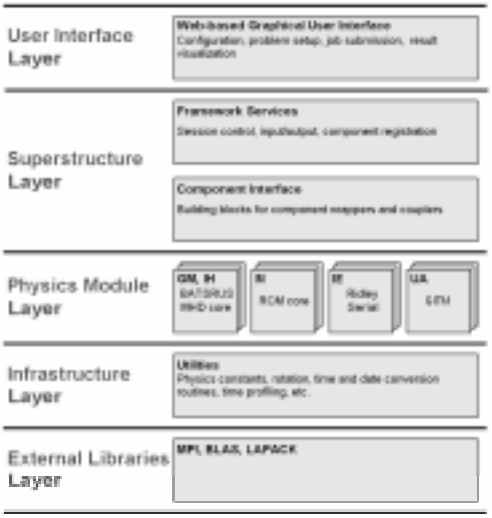
- 2D or 3D block adaptive grid
- 2D or 3D structured grid

Structured grids include uniform and non-uniform spherical and Cartesian grids.

The toolkit obtains the grid descriptors from the components at the beginning of the run. The grid descriptor defines the geometry and parallel distribution of the grid. At the time of coupling the receiving component requests a number of data values at specified locations of the receiving grid (for example all grid points at one of the boundaries). The geometric locations are transformed, sometimes mapped, to the grid of the provider component. Based on the grid descriptor of the provider component, the data values are interpolated to the requested locations and sent back to the requesting component. The interpolation weights and the MPI communication patterns are calculated in advance and saved into a 'router' for sake of efficiency. The routers are updated only if one of the grids has changed (e.g. due to grid adaptation) or when the mapping between the two components has changed (e.g. due to the rotation of one grid relative to the other). In certain cases the coupling is achieved via an intermediate grid, which is stored by the receiving component, but its structure is based on the providing component. The intermediate grid can be described to CON the same way as the base grid of the receiving component.

The framework's layered architecture is shown in Figure 3. The Framework Services consist of software units (classes), which implement component registration, session control, and input/output operations of initial parameters. The Infrastructure consists of utilities, which define physics constants and different coordinate systems, time and data conversion routines, time profiling routines and other lower level routines. The Superstructure Layer, Physics Module Layer, and Infrastructure Layer constitute the "sandwich-like" architecture similar to the Earth System Modeling Framework (ESMF) [15].

Figure 3. The layered architecture of the SWMF.



The SWMF will also contain a web-based Graphical User Interface, which is not part of the ESMF design.

3. Some preliminary results

We present preliminary simulations, which involve all five components of the SWMF prototype. Two

results of simulation are shown in Figure 4 and Figure 5 for the illustrative purposes.

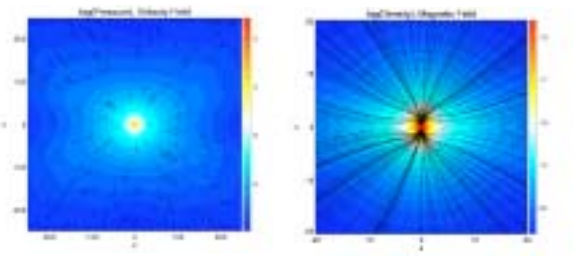


Figure 4. The steady state solution obtained by the Inner Heliosphere model. The left panel depicts the overall pressure and velocity fields, while the right panel shows the magnetic field and density in the vicinity of the Sun. The distance units of the axes are shown in solar radii.

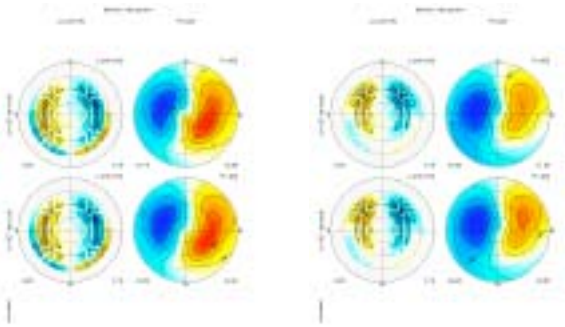


Figure 5. The field aligned current and the electric potential in the ionosphere at the beginning (left panel) and at the end (right panel) of the 1-hour simulation.

The first simulation results proved the importance of loose coupling and advantages of using the SWMF over non-framework programs like BATSRUS. The SWMF is not only more flexible, but in some cases more efficient, than a highly integrated code. For example, when only the GM and IE components are used, the performance depends to a large extent on the execution model. The SWMF allows the concurrent execution of the two components. While IE solves for the electric potential on 2 processors, the GM code can proceed with the MHD simulation on the rest of the processors. As a result, the IE solution is applied to the inner boundary of GM with a small time shift, but this is an acceptable approximation. In this concurrent execution model, the SWMF can run almost twice as fast as BATSRUS.

4. The Web-Based Graphical User Interface

The SWMF web-based Graphical User Interface (GUI) has been developed to provide a visual environment, in which to conduct science runs, facilitating construction of an executable from appropriate components of the SWMF, batch job script preparation for job management system like PBS or LSF, parameter and processor layout files creation, parameter file verification, and, eventually, job submission to a remote high-performance computing host. The high-level architecture of the SWMF GUI, which is shown in Figure 6, is rather standard. It includes a web client (browser), a web server and high-performance computing resources at the remote computing sites. The user interacts with the application through the web client (browser), which sends requests and updates to the web server. The web server has access to multiple high performance computing systems to which it can submit batch jobs and monitor the progress of simulation. The SWMF executables run on these high performance systems.

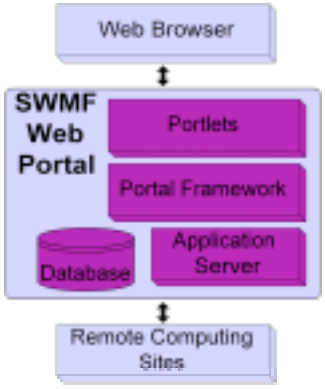


Figure 6. The SWMF Web GUI Architecture.

The GUI has a modular design centered on "portlets" of functionality that plug into a portal framework (see Figure 7). The portal framework that we selected to use is Jetspeed, which is an Open Source, Apache framework from the Java Community. Jetspeed is a reference implementation of Java Standardization Request 168 (JSR 168), making it a suitable platform on which to build the GUI. JSR 168 provides a standard for interoperability between portlets and portals.

Figure 7. The portal web page with portlets.

Each portlet is responsible for accessing content from its source (for example, a Web site, database, or email server) and transforming the content so that it can be rendered to the client. In addition, a portlet might be responsible for providing application logic or storing information associated with a particular user. The portal server provides a framework of services to make the task of writing and managing portlets easier.

From a user's perspective, a portlet is a window in the portal that provides a specific service or information. From an application development perspective, portlets are pluggable modules that are designed to run inside a portal server. The portal framework (portal server) provides a runtime environment in which portlets are instantiated, used, and finally destroyed. Portlets rely on the portal infrastructure to access user profile information, participate in window and action events, communicate with other portlets, access remote content, lookup credentials, and to store persistent data.

The SWMF GUI uses the open-source relational database PostgreSQL to store security information (users, groups, roles), as well as properties of domain objects (parameter sets, execution hosts, executables, and remote queues).

Thus the SWMF GUI implementation follows the simple policy: (1) the use of only open-source yet standardized software, (2) features are added as portlets to the main portal, (3) portlets help divide complex applications into tasks: one group of closely related tasks equals one portlet, and (4) the new portlets can be plugged in as the new features are needed.

5. Conclusions

The In this paper we described the component architecture of the SWMF. The important characteristics of the SWMF are:

1. The SWMF comprises a series of interoperating models of the Sun-Earth system.
2. The SWMF parallel communications are based on the MPI standard. In its current implementation the SWMF builds a single executable.
3. The SWMF contains utilities and data structures for creating model components and coupling them.
4. A component is created from the user-supplied physics code by adding a wrapper, which provides the control functions, and a coupling interface, which performs the data exchange with other components.
5. Each application instance created by the SWMF contains a single Control Module, which controls initialization and execution of the components and is responsible for component registration, processor layout for each component, and coupling schedules.
6. The framework allows a subset of the physics components to execute and can incorporate several different models for the same physics domain.

6. Acknowledgments

We would like to express our gratitude to the NASA Goddard Space Flight Center (GSFC), National Center for Atmospheric Research (NCAR), Rice University and the University of Arizona for collaboration on this project. Our special thanks to J. Kota from the University of Arizona, A. Chan, D. Wolf, S. Sazykin and Bin Yu from Rice University, Cecelia Deluca from NCAR, J. Fisher from NASA GSFC, J. W. Larson and R. Jacob from Argonne National Laboratory, and to the Earth System Modeling Framework (ESMF) Joint Specification Team. The SWMF project is funded by the NASA Earth Science Technology Office (the NASA CAN NCC5-614 grant). G. Toth is partially supported by the Hungarian Science Foundation (OTKA grant T047042).

7. References

[1] N. C. Maynard, Space weather prediction, U.S. National Report to IUGG, 1991-1994, *Reviews of Geophysics*, Vol. 33 Supplement 1995 (Also available at http://www.agu.org/revgeophys/maynar01/maynar01.html).

[2] Powell, K.G., P.L. Roe, T.J. Linde, T.I. Gombosi, and D.L. De Zeeuw, A solution-adaptive upwind scheme for ideal magnetohydrodynamics, J. Comp. Phys., 154, 284, 1999.

[3] Gombosi, T. I., D. L. De Zeeuw, C. P. Groth, K. G. Powell, C. R. Clauer, and P. Song, From Sun to Earth: Multiscale MHD Simulations of Space Weather, in Space Weather, edited by P. Song, H. J. Singer, and G. L. Siscoe, vol. 125, pp. 169–176, AGU, 2001.

[4] Ridley, A. J., D.L. De Zeeuw, T. I. Gombosi, and K.G. Powell, Using steady-state MHD results to predict the global state of the magnetosphere-ionosphere system, J. Geophys. Res., 106, 30,067, 2001.

[5] Ridley, A. J., T. I. Gombosi, D. L. De Zeeuw, C. R. Clauer, and A. D. Richmond, Ionospheric control of the magnetospheric configuration: Neutral winds, J. Geophys. Res., 108(A8), 1328, 2003.

[6] De Zeeuw, D. L., S. Sazykin, A. Ridley, G. Toth, T. I. Gombosi, K. G. Powell, D. Wolf, Inner Magnetosphere Simulations - Coupling the Michigan MHD Model with the Rice Convection Model, Fall AGU Meeting, San Francisco, 2003.

[7] Gombosi T. I., K. G. Powell, D. L. De Zeeuw, R. Clauer, K. C. Hansen, W. B. Manchester, A. Ridley, I. I. Roussev, I. V. Sokolov, Q. F. Stout, G. Toth, Solution-Adaptive Magnetohydro-dynamics for Space Plasmas: Sun-to-Earth Simulations, Computing in Science and Engineering, Frontiers of Simulation, March/April, p. 14-35, 2004.

[8] Szyperski C., Component Software-Beyond Object-Oriented Programming, Addison Wesley, 1999.

[9] Gamma, E., R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley 1995.

[10] Volberg, O., G. Toth, I. V. Sokolov, A. J. Ridley, T. I. Gombosi, D. L. De Zeeuw, K. C. Hansen, D. R. Chesney, Q. F. Stout, K. G. Powell, K. Kane, R. C. Oehmke, Doing it the SWMF Way: From Separate Space Physics Simulation Programs to The Framework for Space Weather Simulation, Fall AGU Meeting, San Francisco, 2003.

[11] Toth, G., O. Volberg, A. Ridley, Space Weather Modeling Framework Manual, Code version 1.0, Center for Space Environment Modeling, the University of Michigan, Ann Arbor, Michigan, 2003.

[12] Volberg, O., D. R. Chesney, D. L. De Zeeuw, K. C. Hansen, K. Kane, R. Oehmke, A.J. Ridley, I.V. Sokolov, G. Toth, T. Weymouth, Space Weather Modeling Framework: Design Policy for Interoperability, Center for Space Environment Modeling, The University of Michigan, Ann Arbor, Michigan, 2002

[13] Edjlali, G., A. Sussman, and J. Saltz, Interoperability of Data Parallel Runtime Libraries. In Proceedings of the Eleventh International Parallel Processing Symposium, IEEE Computer Society Press, 1997.

[14] Decyk, V.K, C.D Norton, and B.K. Szymanski, Introduction to Object-Oriented Concepts using Fortran90, UCLA Institute of Plasma and Fusion Research Report PPG-1560, July 1996. See also the web site: http://www.cs.rpi.edu/~szymanski/oof90.html.

[15] Hill, C., C. DeLuca, V. Balaji, M. Suarez, A. da Silva, and the ESMF Joint Specification Team, The Architecture of the Earth System Modeling Framework, Computing in Science and Engineering, Volume 6, Number 1, 2004.