Towards a Software Framework for Model Interoperability

Sergiu Dascalu, Eric Fritzinger, Sohei Okamoto, Fred Harris
College of Engineering, University of Nevada, Reno
{dascalus, ericf, okamoto, fredh}@cse.unr.edu

***Abstract*-Modern mathematical models for simulation of various systems are becoming increasingly complex and intricate. Since no one model can simulate every aspect of a system, the need for these models to be broken into their component parts is imperative for accuracy and maintainability. However, no model can operate alone without data from an outside source, and so further efforts in the Model Interoperability area of research are necessary. Model Interoperability is dedicated to finding methods with which to couple two or more models. This coupling would allow the models to utilize each other's data to produce more comprehensive and more accurate results. There are several methods for model interoperability, including monolithic, component, scheduled, and communication. Also, there are solutions to the problem of model coupling that use one or more of those methods, but most of them require significant code modification. The goal of the software framework proposed in this paper is to minimize the need for code modification and creation, provide a user interface through which to couple the models, and establish a library of models and other activities that the users of the system can utilize for their own model coupling needs.**

I. INTRODUCTION

The software framework described in this paper, tentatively named Demeter, is part of a climate change research and development project funded by a grant from NSF EPSCoR. There are many solutions currently in use for model coupling, most of which require the user to have intimate understanding of the source code of each model that the user wishes to couple. This tends to lead to delays and even questions as to the validity of the coupled models' output, due to significant source code modification. As a result, while the coupled models have the potential of producing good results, they might also become a system that is difficult to maintain and update.

The goal of the proposed Demeter software framework is to allow scientists, mathematicians, programmers, and other potential users of the system to register new models, use other people's models, link models together in various workflows, and execute these workflows with minimal or no code modification or creation necessary. In addition, it aims to provide a library of models, hosted on remote machines, for the users to access and utilize for their research. This gives the users a broader base of models to choose from, and a cross-platform program with which to visually couple the models together for data transfer.

The paper, in its remaining part, is organized as follows. Section II surveys several methods of model coupling. Section III discusses the existing work in the field of model interoperability. Section IV describes the overall architecture of the Demeter system and provides explanations of the various concepts used in its design. Section V presents the Silverlight Client, the Workflow Runtime Host, and the Server Cluster of the proposed system. Section VI reports on the current progress and outlines planned future work. Finally, Section VII presents the conclusion of this paper.

II. MODEL COUPLING METHODS

Model coupling is a difficult problem with several solutions available [1]. The *monolithic* method assumes the user has both models' source code and attempts to tightly couple them into a single program. The *component* method allows the user to create or modify the models in such a way that they can be swapped out of a modeling system without the need for much, if any, coupling code modification. The *scheduled* method keeps each model as separate programs and the researcher creates the coupling code that takes the output of the first model, massages the data from the output to be suitable for input to the second model, and feeds it into the second model's input. Our previous work has employed this method in the development of scenario-based visual tool to facilitate model execution and data transformation [2]. The last method, *communication*, affords the user the most streamlined model coupling by allowing the models to run concurrently and send messages between them as a means of sharing data [1].

Most of the model coupling methods, however, are mired in the need to modify source code in order to produce a more sophisticated coupling. There are many frameworks available that allow the researcher to use one method or another in a more standardized fashion, but few solutions are fit for all models or model policies. For instance, there may be a scientist that needs to use another researcher's model, but that researcher is unable to share either the source code or the compiled program. This could be for a variety of reasons, from NDAs (non-disclosure agreements) to proprietary information. In this instance, the scientist has no possibility of modifying the source code or using the executable.

However, with the Demeter software framework proposed here, the scientist could simply ask the researcher, who has registered their model with this system, for permission to use his or her model without needing the source code or the executable. This would allow the scientists to continue working by using the model they needed as if they had it without actually having it. This powerful feature will enable

978-1-4577-0434-5/11/$26.00 ©2011 IEEE

researchers to share their work as they see fit without actually giving its associated source code to people.

III. EXISTING WORK

There are many solutions to model interoperability, and each uses a different method, as described in Section II. However, there are a few solutions that seem to stand above the rest when it comes to popular use and standardization. These solutions use the component-based method, which allows for cleaner interoperability code without having to modify the model source code as much as the monolithic or communication-based methods [1].

The Earth System Modeling Framework (ESMF) [3] is a powerful component-based framework that is used to increased software interoperability in scientific applications. It creates a common framework that individual components can be plugged into. It makes a distinction between what are called "gridded" components and "coupler" components. It is cross-platform compatible and uses the Fortran and C programming languages. It is capable of handling parallel execution and has many tools available to the programmer for use. The individual components must be modified to fit within the structure of the framework, and the components also have to be aware of and use the features of the framework, thereby limiting the components' ability to be used in other frameworks.

The Common Component Architecture (CCA) [4] is another component-based method of model interoperability. However, instead of providing a framework, CCA provides a definition language and interface for creating a framework. The Scientific Interface Definition Language (SIDL) is a meta-language that provides an interface definition that is not specific to any language. Each component of a CCA-compliant framework defines a SIDL of what input or output ports it supplies, and the ports can be represented by scalars, arrays, or functions. CCA allows a model coupling programmer to define a framework for coupling other models with their model and, with some standardization, it can be used to couple models on a mass scale. However, it is necessary to create a model with CCA in mind, or wrap the model in a CCA interface. Regardless, the component gets compiled with a program, and cannot be changed without a configuration file alteration, or recompilation.

OpenMI [5] is an interface standard that was produced for component interoperability by the OpenMI Association. It is not a framework or code interface, but rather a standard for code interfaces and the methods that need to be supported to comply with the OpenMI standard. The standard defines methods for initializing a component, retrieving data, creating links, and cleaning up the components. The OpenMI standard affords component writers the ability to create a component without consideration of the programming language, or even the system that it will be plugged into. It allows the programmer total freedom within the bounds of the standard.

Many of the issues with these existing solutions stem from their inflexibility in language support and dynamic replacement. OpenMI seems to stand out as a broader solution to the problem of model interoperability, but it is a standard, not a framework for linking the components themselves.

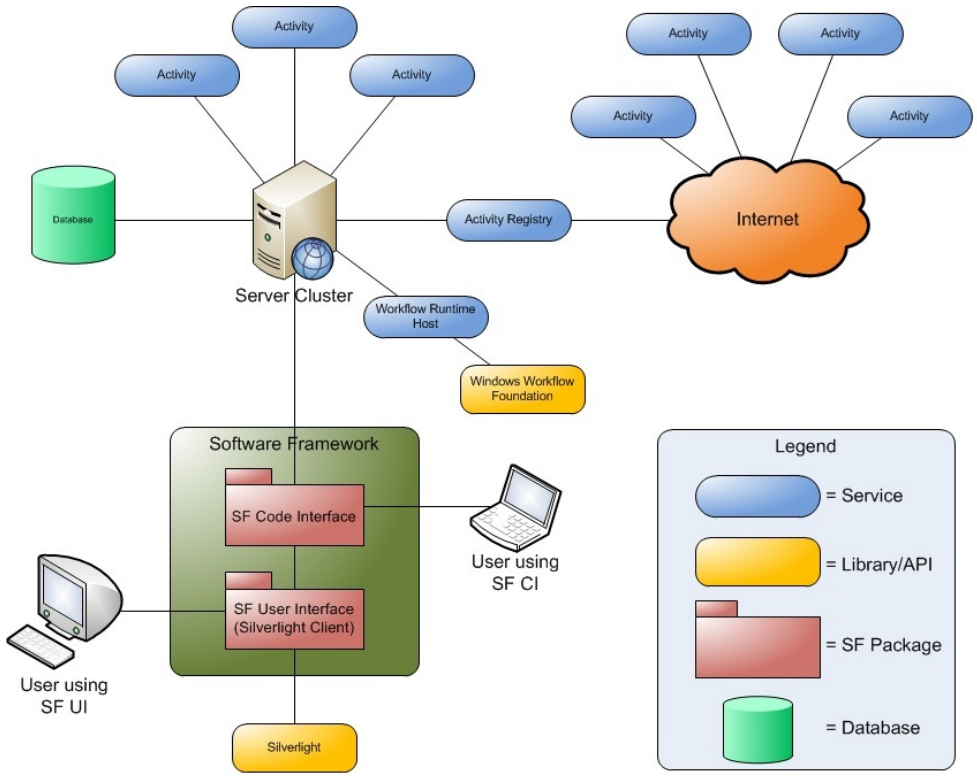

Fig. 1. Software Framework Architecture.

This is where we hope the solution we are proposing will stand out, as it will not only allow these components to link together, but the Demeter software framework will offer several distinct features that are unavailable in most other frameworks.

IV. ARCHITECTURE

The proposed system is comprised of several distinct parts, some of which can be used independently without the others. In designing this system, it was intended to give the users enough freedom as to do what they want, but at the same time it would not give so much freedom that the users would not know what to do. There are three major components to the Demeter software framework that should be noted: the Silverlight Client, the Workflow Runtime Host, and the Server Cluster. There is a small amount of overlap between the components, but the users are given the freedom to only use the parts they need, or want, for their research.

The system makes significant use of web services. A web service is a collection of endpoints that can receive messages from different clients to perform specific functions on the host machine. This functionality is similar to a library in programming, except that the service is at a remote location, hosted on a server. This would, for instance, allow a programmer to access functionality that requires HPC (high-powered computing) resources from his or her local machine without taking up the limited resources of that local machine. So, the user could run a complex simulation model on a server from his or her local machine using a web service, assuming the model was set up to act as a service.

Fig. 1 shows the Demeter framework's architecture as a whole. The legend on the bottom right of the figure shows the meanings behind each of the different shapes. The Demeter software framework itself encompasses two main components, the User Interface and the Code Interface. The User Interface consists of the Silverlight Client, which utilizes the Code Interface. The user may choose to bypass the Silverlight Client altogether, but it is there for the user's convenience. The blue icons indicate services, also known as "activities" in the context of the framework. These activities can be models, data transformers, file converters, and other pieces of code that can be used when linking models together.

The Demeter software framework takes advantage of the web service concept and creates the ability to access the various models as web services to be executed on the HPC nodes that are set up as part of the Server Cluster. The models are coupled together using a workflow metaphor, which is defined using the Silverlight Client. In addition, the software framework allows the users to register new activities/models from their own servers, and although it is not illustrated in Fig. 1, activities on their local machines can be registered as well. When executed, these activities operate seamlessly as part of the Workflow Runtime Host, which is also hosted on the Server Cluster. The Workflow Runtime Host executes the workflow created by the user, and due to it being hosted elsewhere, the user does not need to remain connected.

There is also an API that allows the users to plug their model or activity into the system, and allow others to use it as a means of sharing their work without breaking many of the policies that may be infringed by doing this. This also affords researchers with limited or no HPC resources to utilize other HPC resources as a means to complete their own work. This is the crux of the design for the proposed software framework – to allow for extended sharing of resources for the advancement of research.

V. THE SOFTWARE FRAMEWORK COMPONENTS

The Demeter software framework is divided into three major components: the Silverlight Client, a Workflow Runtime Host, and the Server Cluster that contains the models maintained on site. These three major components, combined together, allow the user to define workflows, use the models maintained on the server, and run them as part of a workflow. As previously mentioned, the user does not need to utilize all three components in order to use the system, but they are closely related and work best as parts of the whole.

1. The Silverlight Client

The Silverlight Client is the first component to discuss, as

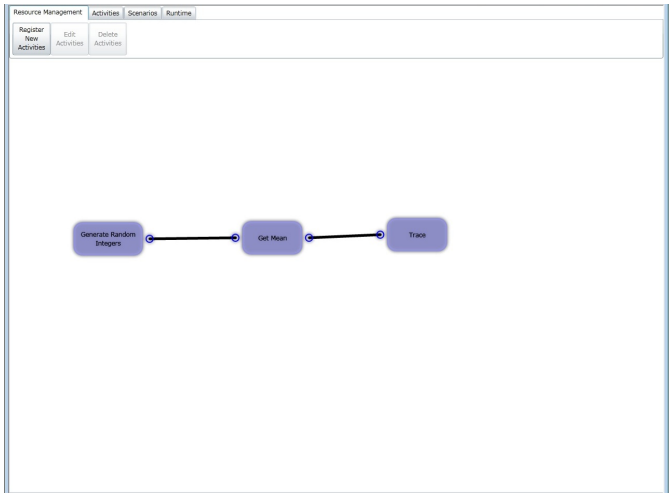

Fig. 2. The Silverlight Client.

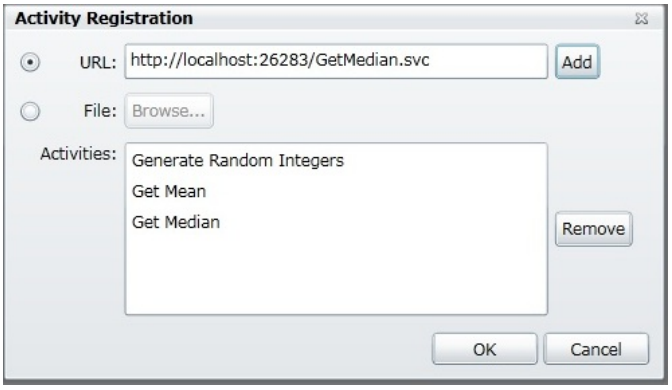

Fig. 3. Activity Registration Window.

it is what allows the user to define workflows, register activities, and send a message to run one of the workflows that were defined. The application itself it embedded into a webpage and the user interface was developed in Silverlight [6], which is cross-platform and cross-browser compatible. Developing the Silverlight Client in this way allows it to be reached by a broader group of people, and the user interface will be displayed consistently among those who use it. Finally, Silverlight applications will also immediately update whenever the user goes to the website that is embedding the application, so there is little need to maintain knowledge of several previous versions, only about the latest one being necessary.

The Silverlight Client shown in Fig. 2 allows the user to define a workflow by linking an activity's output to another activity's input. Represented here is a simple workflow that uses a series of activities. It generates random integers, finds the mean, and displays the output to a trace window after execution of the workflow. While this example does not convey the complexity of model coupling, the software client does allow the user to make the workflow as simple or complex as they wish. Users have the ability to add new activities to the workflow via an activity registration window, brought up by the "Register New Activities" button in the top-left of the screen.

With the Silverlight Client, the user is able to add services or local DLL files as activities to the Silverlight Client. Shown in Fig. 3 is the activity registration window, which shows the "Get Median" activity being added to a list of other registered activities. The idea is to keep this aspect of the Silverlight Client as easy-to-use as possible, allowing the user to more efficiently get started using the workflow editor section of the application.

Activities are added only if they are consistent with the standards of the Silverlight Client. First, the programmer has to implement the activity code interface supplied with the API. The code interface is intended to be simple, but nevertheless to allow the user to create depth to their activities. For instance, the programmer can display an edit-time interface to the user, which allows him or her to setup any parameters that the activity needs before execution. Secondly, the programmer must remain consistent in how they name their inputs and outputs. Each input and output (the small circles attached to the activities shown in Fig. 1) needs to have its own unique name, and when setting up the data outputs, each one must send information out at the end of the activity run. Finally, the user must bear in mind the security restrictions of Silverlight. The Silverlight API does not allow the programmer to manipulate files using code without some kind of user-initiated activity (e.g., clicking a button). Directory and file information is unavailable, and so the programmer must allow for the manipulation of files through the input and output connections of the activity.

Fig. 4 above illustrates a more complex workflow that has completed execution. Seen here is the workflow shown in Fig. 2, but with added activities, and an output to the trace window. Now, instead of merely finding the mean, it finds the median, which requires sorting the data set, and also gets the standard deviation, which requires the mean and the data set. The activities are all executed in a non-deterministic

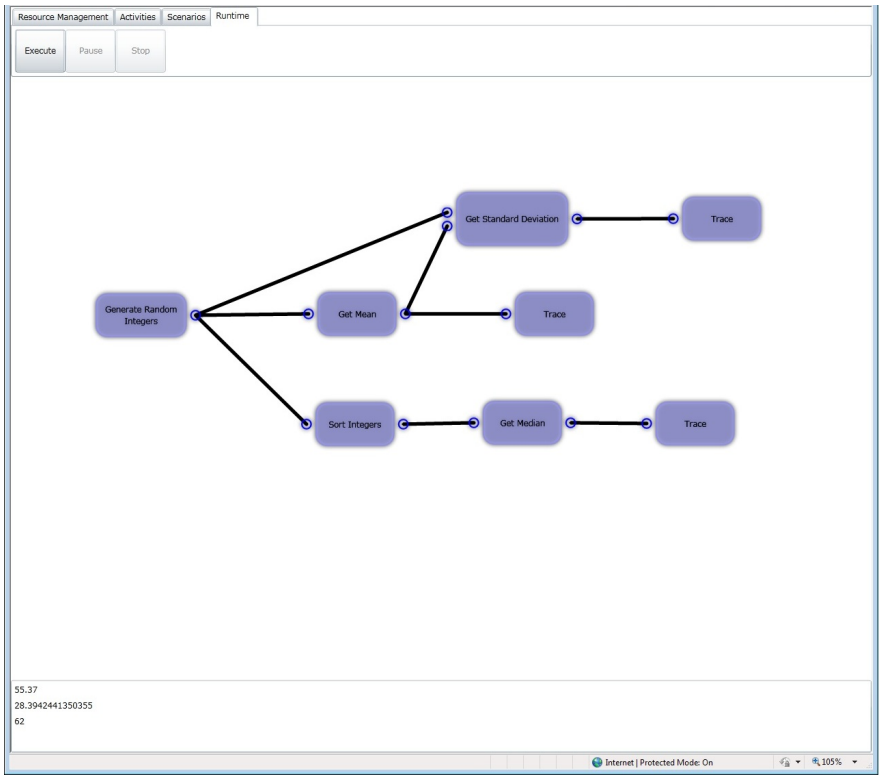

Fig. 4. Completed Workflow.

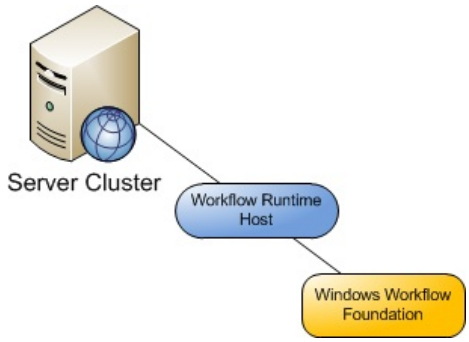

Fig. 5. The Workflow Runtime Host.

fashion, allowing whichever activity is available to execute when they have received all of their input. For example, "Get Standard Deviation" cannot execute until "Generate Random Integers" and "Get Mean" completed execution.

In addition, each of the activities in Fig. 4 comes from different sources. Generate Random Integers, Get Mean, and Get Median all come from their own individual web services. Sort Integers, and Get Standard Deviation come from a single local activity library where both activities were defined. Finally, the Trace activity is built-in to the Silverlight Client. This should illustrate that even though the activities were diverse in their origins, they were all able to share data and be executed.

2. Workflow Runtime Host

The Workflow Runtime Host, presented in Fig. 5, is a web service that is hosted on part of the Server Cluster. It is a relatively small component, but very powerful. It is based off the Windows Workflow Foundation [7] in both terminology and its architecture. The Workflow Runtime Host uses WF as the workflow runtime engine and offers additional features, which are still being developed.

Since the Workflow Runtime Host is a web service, it can be called to execute a workflow, and the user can then immediately disconnect after the execution has begun. The Workflow Runtime Host will continue execution of the workflow and notify the user of any progress or messages it receives via e-mail, text message, or other medium of communication. The user can even log in from a different computer at a later time, open the Silverlight Client, and visually see the progress of the executing workflow.

Another powerful feature of the Workflow Runtime Host is the ability to pause, stop, or resume execution of the workflow. Most importantly, the Workflow Runtime Host can save its progress and keep, temporarily, as much information as possible. This allows the user to resume an execution where something went awry. With as long as the models run, this is one of the more important features to have. While the information cannot be stored indefinitely, the ability to resume it will be there.

Since the Workflow Runtime Host is a web service, anyone with permission to use it will be able to use it as an independent component. This means that programmers will be able to upload their own defined workflows without using the Silverlight Client. This is important because the programmer may wish to use the Workflow Runtime Host and several activities on the Server Cluster, but the Silverlight Client does not have the features he or she needs in order to couple his or her models.

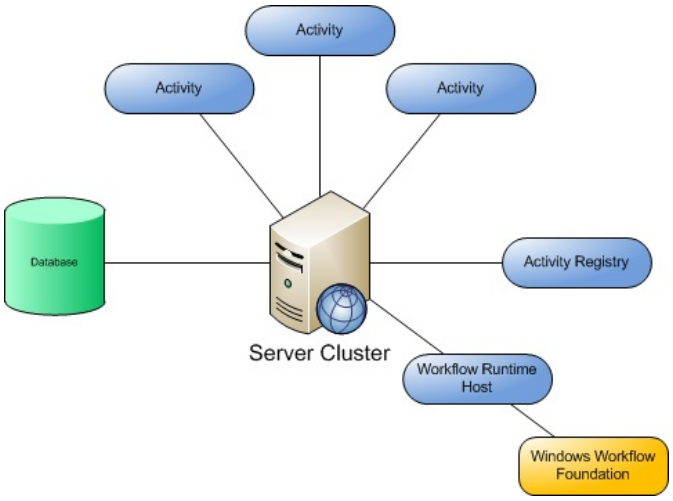

Fig. 6. The Server Cluster.

3. The Server Cluster

The Server Cluster, shown in Fig. 6, is where the activities and the Workflow Runtime Host are maintained. The servers themselves offer powerful computing resources as well as high storage capacity. The reason the Workflow Runtime Host is kept here is because when storing execution data for crash recovery, a large amount of space may be needed. That is where the Server Cluster comes into play.

Several models are going to be kept on the cluster, available to be executed by those who have permission to do so. Each of these activities will be implemented using Windows Communication Foundation (WCF) [8]. This centralized location for the models is the key, as it allows the researchers on this project to access the same models on the HPC resources designed to handle the load. Furthermore, the models will have the ability to store the output to the database for further analysis and visualization.

In addition, the server cluster will maintain a list of registered models that are added by other users. These activities will be available only to those granted access to the activity by the activity's owner. This means that the activity owner will be able to make it publicly available, available only to several users of the system, or available only to the owner. The only information stored in the activity registry is the URL of the activity web service, the type of Internet binding used, and the name and description of the activity.

While the Server Cluster does not serve an active role as a component of the Demeter software framework, it is a crucial component nonetheless. It allows models to be stored and run, the Workflow Runtime Host to execute workflows independently, stores a list of registered activities, and provides high storage capacity.

VI. Current and Future Work

The research, design and development on the Demeter software framework started in early 2010. Significant amount of time was needed to research the field and determine an approach to the problem that would prove useful. The result was a design that is both flexible and unique in its use of web services, recent technologies, and powerful activity-sharing capabilities.

At its current state, the Silverlight Client is in its early prototype stages, and will undergo several revisions before final release. It currently has the ability to register activities, define workflows, and execute them using a runtime engine embedded into the Silverlight Client. Much of the future work will continue to involve developing the complete set of features needed for its initial release, and collaborating significantly with others on the project.

The Workflow Runtime Host is currently developed only in part. It is not a web service, but rather a part of the Silverlight Client that operates solely on the user's machine. It will be extracted to a web service after further research and development on the Silverlight Client is completed.

One important feature to be added later to the Silverlight Client and Workflow Runtime Host is the ability to manage model time and space steps, accounting for the variations used by the different models. For instance, if global climate model A simulates at a 25 km resolution over 10 years on a 6 hour time step, and hydrological model B simulates on a less than 1 km resolution for 2 years on a 1 hour time step, then model A and model B will have to resolve their spatial and temporal differences somehow, likely using an intermediary such as the Silverlight Client and Workflow Runtime Host. How this will be done needs to be researched, but it would be an important addition to the feature set of the software framework.

Finally, the most important part of future work is to motivate the researchers to add their models and other activities to the Demeter software framework. This community effort to share work is what will make the proposed framework stand apart from many other solutions to the model coupling problems. It is important to the process of model coupling that models be available for use by the Demeter software framework's user base. Notably, within the designed setup of the framework model creators can share their work without directly giving their models away.

VII. Conclusion

The field of model interoperability is complex, and coupling models can be an arduous task characterized by significant of code modification and difficulty in obtaining the models necessary for continued research. The Demeter software framework proposed in this paper offers an alternative that limits the need for code modification and at the same time provides a library of models and other activities available to the users. While currently it is in its early stages of development, significant collaboration and research is going into this project from various sources, mostly from climate modelers, but from other sources as well. We hope that this software framework can be a powerful tool for model coupling, and a motivation for the creation and sharing of work designed for the field of model interoperability.

Acknowledgments

This work was made possible through the support provided by the National Science Foundation under Cooperative Agreement No. EPS-0814372. Authors would like to thank Dr. Dan Ames, from Idaho State University, who suggested the use of web services for the proposed software framework for model interoperability.

References

[1] T. Bulatewicz, "Support for model coupling: An interface-based approach," PhD dissertation, University of Oregon, 2006.

[2] S. Okamoto, E. Fritzinger, S. Dascalu, F.C. Harris, Jr., S. Latifi, and M. McMahon, Jr., "Towards an Intelligent Software Tool for Enhanced Model Interoperability in Climate Change Research," in *Proceedings of the 2010 World Automation Congress (WAC-2010)*, Kobe, Japan, IEEE Computer Society, pp. 1/1-6.

[3] Earth System Modeling Framework; "ESMF"; http://www.earthsystemmodeling.org/; Accessed February 18, 2010.

[4] CCA Forum; "The Common Component Architecture Forum"; http://www.cca-forum.org/; Accessed February 18, 2010.

[5] OpenMI Association: "OpenMI Association"; http://www.openmi.org/; Accessed October 1, 2010.

[6] Microsoft; "Home: The Official Microsoft Silverlight Site"; http://www.silverlight.net/; Accessed June 15, 2010.

[7] Microsoft; "Windows Workflow Foundation"; http://msdn.microsoft.com/en-us/netframework/aa663328.aspx; Accessed June 15, 2010.

[8] Microsoft; "Windows Communication Foundation"; http://msdn.microsoft.com/en-us/netframework/aa663324.aspx; Accessed June 15, 2010.