International Environmental Modelling and Software Society (iEMSs)
2010 International Congress on Environmental Modelling and Software
Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada
David A. Swayne, Wanhong Yang, A.A. Voinov, A. Rizzoli, T. Filatova (Eds.)
http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings

Methods to Register Models and Input/Output Parameters for Integrated Modeling

James G. Droppo[a], Gene Whelan[b], Michael E. Tryby[c], Mitchell A. Pelton[d], Randel Y. Taira[e], and Kevin E. Dorow[f]

[a]Pacific Northwest National Laboratory, Richland, Washington 99354 (James.Droppo@pnl.gov); [b]U.S. Environmental Protection Agency, Athens, Georgia 30606 (Whelan.Gene@epa.gov); [c]U.S. Environmental Protection Agency, Athens, Georgia 30606 (Tryby.Michael@epa.gov); [d]Pacific Northwest National Laboratory, Richland, Washington 99354 (Mitch.Pelton @pnl.gov); [e]Pacific Northwest National Laboratory, Richland, Washington 99354 (Randel.Taira@pnl.gov; [f]Pacific Northwest National Laboratory, Richland, Washington 99354 (Kevin.Dorow@pnl.gov)

Abstract: Significant resources can be required when constructing integrated modeling systems. In a typical application, components (e.g., models and databases) created by different developers are assimilated, requiring the framework's functionality to bridge the gap between the user's knowledge of the components being linked. The framework, therefore, needs the capability to assimilate a wide range of model-specific input/output requirements as well as their associated assumptions and constraints. The process of assimilating such disparate components into an integrated modeling framework varies in complexity and difficulty. Several factors influence the relative ease of assimilating components, including, but not limited to, familiarity with the components being assimilated, familiarity with the framework and its tools that support the assimilation process, level of documentation associated with the components and the framework, and design structure of the components and framework. This initial effort reviews different approaches for assimilating models and their model-specific input/output requirements by considering four case study examples: 1) modifying component models to directly communicate with the framework (i.e., through an Application Programming Interface), 2) developing model-specific external wrappers such that no component model modifications are required, 3) using parsing tools to visually map pre-existing input/output files, and 4) describing and linking models as dynamic link libraries. Most of these examples are derived from assimilation efforts for the widely distributed modeling system called Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES). The review concludes that each has its strengths and weakness, the factors that determine which approaches work best in a given application.

Keywords: Integrated Modeling; Legacy Model; Data Wrapper; Model Linkage; Data Transfer; Database; Multimedia Modeling

1. INTRODUCTION

Integrated modeling systems combine model and database components into a single modeling framework. Often both new and old components are included. Both time and cost can be significantly reduced with well-vetted "off-the-shelf" models. A framework designer faces the challenge of including capabilities that will meet the assimilation needs for the components to be used in the framework. The trade-off is between investing more in the framework model assimilation capabilities and requiring more resources when constructing an integrated modeling system within a framework. This paper addresses assimilation approaches used for legacy components and new externally developed components.

It is potentially costly and labor-intensive to construct an integrated modeling system within a framework. A typical development effort requires assimilating dissimilar components (e.g., models and databases) created by different developers, often using a different software language or database system. The framework's functionality needs to allow the framework's user to assimilate components often based on a limited understanding of the details of a component's characteristics. The framework needs to provide the framework's user with the ability to assimilate a wide range of component-specific input/output requirements in a manner that accounts for their associated assumptions and constraints. The challenge of assimilating disparate components into an integrated modeling framework varies in complexity and difficulty. A key factor that can be a showstopper is the compatibility of the spatial and temporal design of the framework with the external component. Other major factors can be the user's familiarity with the components and framework along with the level of component documentation available for components. The capabilities of the framework component assimilation tools need to match the input/output functionalities of the components being assimilated.

This paper addresses some processes that can be used for model assimilation with a framework. The discussion is based on the experience of the authors in model assimilation efforts for integrated modeling systems. Most of these approaches are illustrated using the widely distributed modeling system called Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES). In FRAMES, logical groups of parameters and metadata are formally defined as DICtionary (DIC) files. These files, which are used by the FRAMES application programmers interface (API) to store and obtain data, have a highly organized, structured design [Gelston et al. 2004].

2. MODEL ASSIMILATION APPROACHES

A major challenge in the development of a functional integrated modeling framework is being able to design an effective system for incorporating models. There is, of course, no one best approach. Instead, there are a variety of approaches that have various strengths and weaknesses. Many different ways are used for registering models and parameters with frameworks. Major considerations for designing a system-specific model incorporation system are related to the origin of models, the approach for linking models, the desired flexibility for model implementations, and limitations related to obtaining the desired level of performance for the integrated modeling system.

A basic tenet of integrated modeling systems is that computations are performed that require the models to transfer data. An output from one model is an input to another model. This data exchange may be accomplished by using files, static databases, or dynamic databases. Whatever data-exchange methods are used, the model must be registered to implement those exchanges of model input and output data. The data that define the connectivity of the models are referred to as model boundary conditions. When integrating (or connecting) two models together, the challenge is in accurately passing output data from the first model such that it can be used as input data for the second model. Even in cases where models are compatible in terms of data being transferred (i.e., parameters and units match), it is extremely rare that the first model's output file format identically matches the input file format of the second model.

3. MODEL REGISTRATION EXAMPLES

Integrated modeling frameworks provide different levels of support for implementing models. The input and output parameters must be defined and registered in the framework before a model can be implemented. As described in Whelan et al. (2010), FRAMES uses a formal parameter definition procedure that automatically matches parameter properties and handles unit conversions. Many of the examples presented below use FRAMES development tools (i.e., sets of subroutines and functions) to communicate with the

FRAMES API. These tools support data transfers, obtaining data properties and other functions in a number of computer languages, and versions of computer languages. These examples also use the FRAMES generic data entry/viewer capability (DCE editor) (Whelan et al. 2010). Models can be registered within frameworks by modifying the code to work within the framework or by creating a model wrapper outside the framework. The first assimilation example modifies the source code to work within the framework. The second two examples create model wrappers either by writing code or using a data-parsing wrapper wizard. The fourth example creates model wrappers to conform to a dynamic link library (DLL) standard.

3.1 Example 1 – Modifying Source Code

This example modifies the model source code to input and output data from a model. These modifications are run as an integral part of the model.

Background

As noted above, when integrating (or connecting) two models together, the challenge is in accurately passing output data from the first model such that it can be used as input data for the second model. A schematic of the implementation and linkage of legacy models in FRAMES is shown in Figure 1. With this approach, a model is modified such that the model can 1) read input data from various data sources and 2) produce output data in a form that can be consumed by downstream models. Note that the downstream module output data source becomes the upstream module input data source for the next linked model.

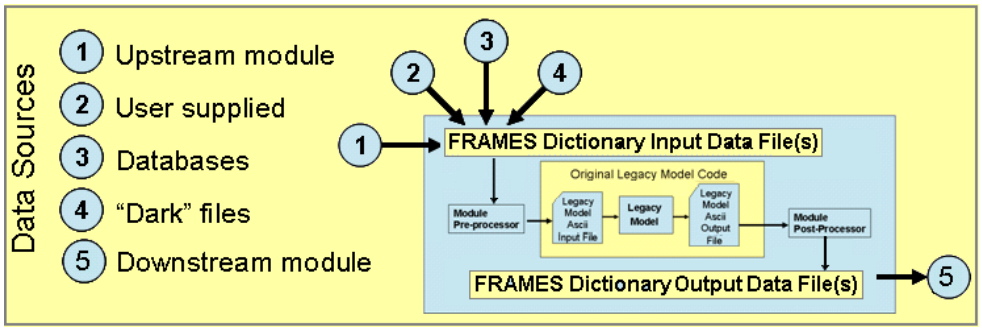

Figure 1. Schematic View of Model Implementation in FRAMES

Application

The U.S. Nuclear Regulatory Commission (NRC) model for codifying potential routine air effluent rates for an operating boiling water reactor (BWR) is referred to as GALE BWR-GE. As part of an effort to update this model whose original development was in the 1970s, a revised FORTRAN source code was developed (Droppo and Pelton 2010). The revised GALE BWR-GE model source code now includes calls to the FRAMES API for the model's 1) input and output data exchanges and 2) initialization of model parameters.

This model implementation in FRAMES was conducted after the development of wrappers for unmodified legacy versions of this model (described below). The process was very straightforward: 1) the read statements for inputs were replaced with API calls to obtain data from dictionary-based input data files and 2) code was added to write the desired output parameters through the FRAMES API to dictionary-based output files. Issues were encountered related to the transfer of data between the model's FORTRAN code and the FRAMES C codes. Typically, additional code was needed to address these issues. Overall, we found the time to implement the model with this approach was much less than

for the wrapper-development approach. We also found the linkages to be cleaner and faster using the model modification approach.

Summary

Modifying the model source code provides a direct link between model and framework. Using this approach requires a good understanding of the selected model. Quality assurance/quality control (QA/QC) testing should be performed on the modified model to verify that the modifications did not introduce errors. The model modification approach has the potential of often being the least labor-intensive model assimilation approach.

3.2 Example 2 – Writing Code to Create Model Wrappers

This approach uses "wrapper" computer programs to feed input data to and obtain output data from a model. These wrapper programs run separately from the model—allowing the model to be used in an essentially unmodified form. In many situations, particularly for legacy models, it is highly desirable to use models in their unmodified state to verify the maintenance of the inherent functionally of a model.

Background

Registering a model with wrappers requires creating wrapper programs and handling the logistics of data transfers and model execution. Typically, pre-run wrapper programs create the input files and/or databases, and a post-run wrapper program imports the model output data from a model output file. When such model wrappers are used on an unmodified model, then the QA/QC testing needs be performed only on the wrappers. Two distinct methods were used to create the model wrappers: 1) write custom source code for each model wrapper and 2) use generic model wrapper programs to automate the model-wrapping process. These tasks are easiest for models with "well-behaved" input and output files—in which locations of needed data can easily be uniquely defined. Writing model-specific wrappers using the FRAMES API calls provides a high level of flexibility in creating the model wrappers. However, this approach can be the most labor intensive of the various wrapper approaches. The user must address the details of model data transfers plus language/complier programming data-transfer issues. Using generic software wrappers can be an effective approach for implementing models. A large part of the model registration process can be quickly performed. Programming-related data-transfer issues can be resolved one time in generic wrapper codes. The use of a generic wrapper program in a new application is limited by its inherent capabilities.

Application

Model wrappers for NRC's reactor emissions codes, the GALE codes, were developed by using a combination of custom and generic model wrapper programs (Droppo and Pelton 2010). A model run involves a series of operations (Figure 2). Step 1 starts with selecting a list of radionuclides to be addressed. Step 2 is to select which GALE code is to be run. In step 3, the user enters the code-specific input data using the FRAMES generic data editor (DCE). These input data are stored in a FRAMES DIC database file. Next, in steps 4 to 7, the wrapper programs and the model are run using batch files. Each of the GALE codes reads a wrapper-created text file for input and produces a text file read by a wrapper to get model results.

Generic wrapper programs were written for mapping the data input files (Pmod) in step 4 and results in the output files (Rmod) in step 7. These wrapper programs exchange data between FRAMES DIC input/output files and flat GALE input/output files. These generic wrapper program codes also were used to implement two other models (NRC GASPAR

and LADTAP-II models). A custom wrapper program was required for matching GALE and FRAMES names for radionuclides (Cmod) in step 6. Step 8 occurs as part of the data input process for the downstream model. In step 9, custom data wrappers were needed for creating the FRAMES DIC database files required by "downstream" air and water models (Dmod). An alternative approach would be to have the transport-pathway specific facility data be part of the facility inputs, so the required downstream FRAMES DIC database files could have been created in Step 7.

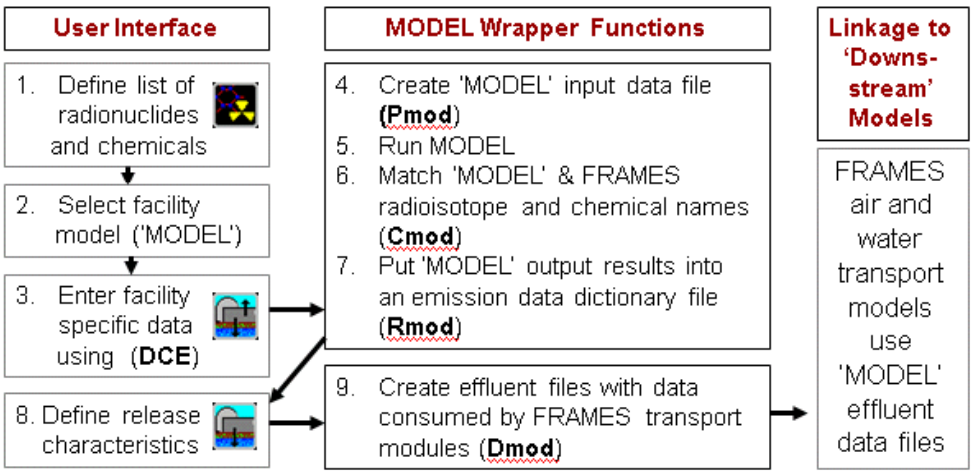

Figure 2. Wrapper-based "MODEL" Implementation of the GALE Codes

Summary

Creating model wrappers worked well for assimilating the GALE codes. Using generic wrapper programs for "well-behaved" data exchanges greatly simplified those efforts. However, the creation of custom wrapper codes for model-specific requirements was quite labor-intensive. The assimilation of unmodified GALE codes greatly reduced QA/QC testing requirements compared to the model-modification approach.

3.3 Example 3 – Assimilation of a Model Using a Data-Parsing Wrapper

This approach emulates the model wrapper approach discussed above using an interactive framework-based development environment for 1) creating model "wrapper" functionalities needed for input/output model data exchanges and 2) handling the logistics of running the model.

Background

Using an interactive data parsing approach can greatly reduce the resources needed to register a model using model wrappers. Assuming that the pertinent model parameters are defined in dictionaries, Dorow et al. (2007) describe parsing techniques used to map the model parameters. The concept is to have the user defining the data-mapping specifications through visual inspections, using a graphical user interface (GUI), of the data files. In practice, the model's files must be in (or converted to) a readable text format.

Application

A data-parsing wizard was developed for a FRAMES application (Dorow et al. 2007) for mapping model output parameters. "Text File Tables" are used to define tables of data

within the output file, “Text Spans” are used to define areas of discrete values, and “Transforms” are used to parse and concatenate data formats such as dates, as necessary.

To register a discrete value, the user defines the exact location of the value by row and column or by some unique identifier that precedes the value (e.g., unique descriptor). Once mapped, any value that shows up in that location is registered with the system and passed along to downstream models requiring that DIC. Tables are more complicated—only a portion of the table might be needed. One must uniquely identify the exact data locations in tables (rows and columns) accounting for shifts in data location that may occur. To address this issue, mapping may reference absolute row numbers or indicate relative row location using unique static information (Dorow et al. 2007).

Summary

The data parsing approach can be used effectively in applications where the location and formats of the data in the mapped files can be uniquely identified. For this approach to work, a successful base run file needs to exist as a template for mapping the data values. The advantage is that once this base run case is registered, new simulations based on this case can be run. The limitation is that a re-mapping and re-naming of the model often will be required to address different runtime options. A major challenge with the data parsing approach is providing the functionality of addressing the many permutations for the content, format, and location of data in model input/output files.

3.4 Example 4 – Model Wrappers to Implement a Linkage Standard

Background

Unlike the linkage cases considered above, the final approach discussed here does not use a framework to manage data exchange; rather, linkage and data exchange occurs directly between model components communicating via a standard.

Application

OpenMI is a software standard that facilitates the linkage of individual models into integrated modeling systems (OpenMI 2010). It defines data structures and protocols for data exchange and has facilities for handling the spatial and temporal mismatch between model domains. The standard is open source and computing-platform independent. OpenMI places the responsibility for implementing its runtime capabilities on the component developer. Thus, the model integration process requires intermediate to advanced software development skills, development of new software, and in some cases, significant revisions of existing computational cores. OpenMI promotes a “Wrapper”' pattern for model integration not unlike the model integration strategy described above.

Converting existing computational cores into OpenMI linkable components requires several well-defined steps. The first step, however, is the most difficult. The existing computational core must be converted from an executable to a DLL and must expose entry points through the following APIs: 1) initializing, running, time-step control, finishing, and cleaning-up, 2) setting initial conditions, and 3) accessing the computational core data model (e.g., setting and getting values for input and output exchange variables). Once the DLL is created, the process of conversion to an OpenMI Linkable component is straightforward.

Summary

OpenMI is an emerging standard; the future implications of which are uncertain. Many organizations have made significant investments developing integrated modeling frameworks that work well for their needs but are non-compliant with this standard. Short of outright adoption of the standard, OpenMI may facilitate integrated modeling by 1) providing a means of creating OpenMI linkable components that can be embedded in existing and new frameworks, 2) provide a common API for linking with integrated modeling tools and making them interchangeable between frameworks, and 3) defining an API for linking disparate integrated frameworks together.

4.0 CONCLUSIONS

In summary, Table 1 lists the general model implementation functionalities that need to be provided. The approaches discussed above illustrate ways of meeting the listed functionalities. In all methods, some level of software programming, compiling, testing, and documentation is normally necessary. The "best" assimilation approach for a given model may be some combination or hybridization of these approaches.

Table 1. Model Implementation Processes

Functionality	Description
1. Definition of Model-Specific Input Data	In addition to data from an "upstream" model, a model often requires that run-time parameters be defined. Implementing systems for handling these model-specific data represents a special challenge for generic model registration systems.
2. Framework Support of Model Import of "Dark Data"	A model often requires definition of run-time input parameters that do not need to be assessed by the modeling framework. These data are referred to as "dark data" because the framework does not provide a means of accessing or modifying these data.
3. Model Access to Framework Databases	It is often necessary to access global framework databases when implementing a model. For example, frameworks often have GIS and constituent property databases.
4. Match Linked Data Parameter Properties	The exchange of a parameter value between models needs to match the data properties (units, time, spatial and temporal average, etc.) from an "upstream model" with the properties of data expected by the "downstream model."
5. Logistics Support for Running the Model	Registering a model requires that the logistics for running the model are in place. In addition to the applicable executable and batch files, the model implementation must include defining the paths and files for the various model data transfers between files and databases as well defining the model run status.
6. Source Code-independent Communication and Data Transfers	Model and framework software are often written in different computer languages. The model registration process must allow for variations in data structures, formatting conventions, and other differences in language procedures.
7. Means of Accessing Model Results	The model registration needs to include definition of the means that the framework will use to access the results generated by a model.

Our experience with FRAMES is that wrapper programs can easily be built to handle "well defined and well behaved" input and output file structures—but the task of developing wrapper programs to cover all possible file structures is prohibitively complex. The data-parsing wizard worked well in its original application. However, it has not found wide use yet in other applications, as was expected. The main impediment has been the need for additional file-mapping capabilities. Although this approach should require no new coding, we have found that in practice, some coding may be required to address model-specific mapping issues such as non-standard constituent naming conventions and non-unique data-mapping locations in model files.

The OpenMI standard is not seen as perfect and will not eliminate all the technical difficulties associated with integrated modeling. The foundation of a standards-based organization for integrated modeling is, however, an important and significant step forward for the integrated modeling community. Each approach considered works well in certain situations and not so well in other situations. Modifying the model source code to exchange data through the framework API is normally the easiest method for assimilating legacy models. Writing model wrappers works well for models whose source codes need to be used in an unmodified form. Creating a model wrapper wizard works well to perform specific functionalities required by a specific application. Creating specifications for implementing a model as a DLL has the potential to provide the best linkage performance.

The views expressed in these Proceedings are those of the individual authors and do not necessarily reflect the views and policies of the United States Environmental Protection Agency. Scientists in EPA have prepared the EPA sections, and those sections have been reviewed in accordance with EPA's peer and administrative review policies and approved for presentation and publication.

5.0 REFERENCES

Dorow, K.E., S.L. Eaton, C.L. Giancola, R.L. Johnson, BD Lawler, R.Y. Taira, and J.L. Kirk. *Integrated Water Resource Modeling System (IWRMS), Model Integration Wizard User Guide*. Version 1.0. PNNL-15877, Rev. 3. Pacific Northwest National Laboratory, Richland, Washington, 2007.

Droppo, J.G., Jr., and M.A. Pelton. *Formalized Linkage of Atmospheric Dispersion Models to Transport in Other Media*. PNNL-SA-70208, American Meteorological Society, 16th Conference on Air Pollution Meteorology, Atlanta, Georgia, 2010.

Gelston, G.M, M.A. Pelton, R.E. Lundgren, K.J. Castleton, G. Whelan, B.L. Hoopes, J.L. Kirk, A.J. Pospical, M.A. Eslinger, J.G. Droppo, Jr., and D.L. Strenge. *Using Dictionaries to Manage Data Within a Modeling Framework System*. PNWD-3507. Battelle—Pacific Northwest Division, Richland, Washington, 2004.

OpenMI. 2010. OpenMI – Home, http://www.openmi.org/openminew/ (last accessed March 13, 2010).

Whelan, G., M.E. Tryby, M.A. Pelton, J.A. Soller, and K.J. Castleton. "Using an Integrated, Multi-disciplinary Framework to Support Quantitative Microbial Risk Assessments." Proceedings of the *2010 International Congress on Environmental Modelling and Software*, D.A. Swayne, W. Yang, A.A. Voinov, A. Rizzoli, and T. Filatova (Eds.), Ottawa, Canada, July 5–8, 2010.