Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

THE POTENTIAL COUPLING INTERFACE: METADATA FOR MODEL COUPLING

Tom Bulatewicz
Janice Cuny
Maureen Warman

Computer and Information Sciences
University of Oregon
Eugene, OR 97403, U.S.A.

ABSTRACT

Model coupling is a nontrivial task that is not adequately supported in existing frameworks. Our long term goal is to support the fast-prototyping of model couplings, enabling scientists to quickly experiment with a variety of linkings without having to make an upfront investment in reprogramming. This paper introduces the centerpiece of our framework, the Potential Coupling Interface (PCI), a visual representation of a model code based on simplified control flow graphs. The PCI serves three roles: it is a new form of metadata describing the coupling potential of a model; it is the vehicle for the specification of couplings; and it is the basis for automatic code generation. It is easy to specify and once specified, it is available for all future coupling activities. The PCI allows scientists to focus on the important domain and model issues of coupling without having to revisit legacy code for each new effort.

1 INTRODUCTION

Scientists often simulate complex physical phenomena by coupling models of simpler subsystems. In our application domain of Hydrology, for example, watershed simulations are composed from interacting ground water, surface water, and solute-transport models. The task of coupling existing models, however, is nontrivial. It occurs on three levels. At the domain level, the scientist must identify an appropriate coupling. At the model level, where the physical systems are represented mathematically, s/he must resolve any incompatibilities regarding, for example, parameterization, data type or resolution, or time step length. At the program level, where the models are represented as code, s/he must integrate the codes, which are often poorly understood legacy codes that were originally written as special purpose, standalone programs. We use the term model coupling to describe this entire process, although most existing work focuses on the model code level.

At the model code level, there are several approaches. The most basic is a brute force merger of the existing codes into a single program (Jobson and Harbaugh 1999, Guo and Langevin 2002). This involves time-consuming reprogramming, requires detailed knowledge of the underlying programs, and does not facilitate reuse. More sophisticated approaches use frameworks designed to support model coupling. We categorize such frameworks as communication-oriented or component-oriented. Communication-oriented frameworks (Valcke et al. 2000, Kauffman and Large 2002, Beckman et al. 1998, Guilyardi et al. 2003, MpCCI 2003, Sydelko et al. 1999, Larson et al. 2001) facilitate the transfer of data between concurrently executing model codes that have been instrumented with appropriate communication calls. These frameworks may provide domain-specific data models, standard data transformations, and parallel data transfer. Their users must have low-level, detailed knowledge of each of the source codes, and the resulting programs are not reusable in general. Component-oriented frameworks (Ford et al. 2003, Piacentini 2002, Hill et al. 2004, Parker et al. 1998) facilitate reuse and ease of programming with plug-and-play type environments. Introducing existing codes into such an environment though, often requires extensive recoding. The existing code must be reorganized into independent components with well-defined interfaces that can be composed by mapping outputs to inputs. This approach works well when the modules are written from scratch but it is often difficult to decompose complex, unstructured legacy codes into simple modules that can interact with straightforward input/output relationships. Both the communication- and the component-oriented frameworks require a significant upfront investment before a scientist can even begin to investigate potential couplings to a new model. We are developing an alternative that combines features of both approaches to support the fast prototyping of coupled models.

Our goal is to enable scientists to easily experiment with a variety of couplings before investing in recoding ef-

forts. Like communication-oriented frameworks, we will support the use of annotated legacy codes rather than recoding. Like component-oriented frameworks, we will promote reuse, allowing the scientist to plug together models that have been made available within the framework. We will support all aspects of model coupling (identifying appropriate models, specifying their interactions, and integrating their codes), while allowing the scientist to work at a high-level of abstraction without becoming enmeshed in low-level source code details. In this paper, we report on the centerpiece of our approach: a high-level, metadata description of the coupling potential of a model, called the Potential Coupling Interface (PCI).

Computational models are a unique class of programs in that they require a great deal of knowledge beyond their source code and user interfaces in order to be used correctly. Thus their documentation may include such things as parameterization, spatial aspects (dimensionality, grid extent, and granularity), temporal aspects (fixed/variable time step, and termination conditions), and model type (time-dependent/independent, discrete event/continuous, etc.). Although this information is necessary, it is not sufficient for coupling. For coupling, the user must know which state variables can be accessed and where those accesses can occur: Can a specific state variable be modified at any point in a simulation, or only during an initialization phase? Can a set of data be read by another model on every iteration of a solution loop, or is it only meaningful after the solution has converged? If two models use each other's state on each time step, how are their time steps to be coordinated? Can a subcomputation that updates a state value be repeatedly executed, or must the full model always be executed from start to finish? The Potential Coupling Interface succinctly captures this relevant information.

The PCI is intended to capture the information about a model code relevant to coupling: it describes the overall structure of the code, and it specifies the locations where specific state variables can be read or modified by another model. It serves as

- a form of model metadata that clearly conveys the code's coupling-relevant aspects,
- a vehicle for graphically specifying the coupling of two or more codes, and
- the basis for automatic generation of coupling code.

Thus, the PCI plays the same role as the interface descriptions used in component frameworks but it is able to capture more complex interactions between programs.

In the next section, we describe the PCI and how it is created. In Sections 3-5, we demonstrate the use of the PCI in each aspect of the coupling process: as metadata; as the vehicle for coupling specification; and as the basis for code generation. In Section 6, we summarize the role of the PCI in our ongoing work.

2 THE POTENTIAL COUPLING INTERFACE

Our aim is to develop a representation that allows all coupling relevant aspects of a model to be readily understood by scientists who are not familiar with its source code, while at the same time maintaining a correspondence with the underlying program that is sufficient for automatic instrumentation and code generation. In order to encourage the participation of the original programmers, we want our representation to be easy to create. To accomplish these goals, we base our representation on control flow graphs (CFGs) in which nodes represent sections of code and directed edges represent the flow of control between them. Work in program comprehension suggests that succinct CFG representations are effective in describing programs (Storey, Fracchia, and Muller 1997). We generate an initial graph directly from the model code and then allow the user to manipulate and simplify that graph with the help of a tool, PCIAssist. We preserve the correspondence to the underlying code through all transformations, so that we can use our CFGs in the automatic generation of coupling code.

To create a PCI, the author or someone familiar with the source code annotates it, marking potential interaction points and state variables that can be read or modified at those points. For example, the code segment

```
      IDGB=0
      COUPLING POINT('Before time step loop',TRB)
C start time-stepped loop
      WRITE(LUOUT,2150)
      DO 100 j=1,NHR
```

(taken from the DAFlow model below) creates and labels a point of potential interaction (COUPLING POINT) where the TRB array can be imported or exported. Once the code is annotated, it is translated from its source language (Fortran/C/C++) into a structured intermediate form using the Program Database Toolkit (PDT) (Mohr et al. 2000). The intermediate form is then parsed to generate a complete control flow graph. Generally these graphs are huge, far too large to be comprehensible. To reduce them, we use a modified form of the graph reductions used in Interval Analysis (Aho and Ullman 1972). The original algorithm simplifies a graph by inspecting the edges of each node, while our algorithm additionally inspects the type of each node (control statement, annotation statement, etc.). This allows our algorithm to preserve annotations and the control structures surrounding them in the reduced graph. All other nodes are collapsed, retaining only the aspects of the graph relevant to coupling.

As an example, consider DAFlow (Jobson 1989), a distributed-parameter, time-dependent, surface-water flow model written in Fortran. It has a simple (and common)

structure: it reads parameters and builds data structures during an initialization phase, and then it executes a single time-stepped loop computing the water levels in each branch segment. DAFlow could potentially interact with a time-dependent, spatial ground-water model in several places. It could, for example, import its parameters or it could import/export simulated water levels on each time step. In the later case, different length time steps would have to be coordinated. Typically this would happen in one of two ways: the simulations could proceed in lock step, or one of them could be invoked for some number of steps during each iteration of the other. To anticipate both of these possibilities, the creator of the PCI would place annotations at the top and bottom of the loop (for lock step execution), and immediately before and after it as well (for multistep execution). The full CFG for DAFlow contained 1082 nodes; it was automatically reduced to the 28 node graph shown in the left of Figure 1, with coupling points marked with a double arrow icon. The reduced graph is the starting point for the PCI creator. The PCIAssist tool supports manipulation of the graph, allowing the programmer to add variables for import/export at interaction points, view related source code statements, annotate nodes with descriptions, add and remove coupling points, further reduce or expand sections of the graph, and adjust the graph layout. Its interface uses JGraph (Alder 2002) and is shown in the bottom right of Figure 1, and the final PCI for DAFlow developed in PCIAssist is shown in the top right. The VASE visualization system (Jablonowski 1993) used CFGs in a similar manner to specify breakpoints for code visualization. VASE graph reduction was not automatic but was specified by the user who demarcated blocks of source code that were to be coalesced; s/he then identified breakpoints by selecting CFG edges.

In ongoing work, we are conducting preliminary usability studies on the PCI, evaluating it both as a convenient form of documentation (Can the developer easily specify a PCI that accurately captures the needed information for a reasonable range of potential, perhaps unanticipated, couplings?) and as a tool for comprehension (Can the user easily interpret the PCI to understand relevant program behavior without resorting to exhaustive analysis of source code?). We are also constructing a set of templates that can be used to ease PCI creation, to make coupling specifications more convenient, and to improve our ability to generate efficient coupling code. Templates capture standard, perhaps domain-specific, model code structures. Thus, for example, DAFlow has a common structure: it reads parameters and builds data structures during an initialization phase, and then it executes a single time-stepped loop. Instead of manipulating the CFG to make this apparent, the PCIAssist user will be able to match coupling points in the CFG with coupling points in the template.

3 THE PCI AS METADATA

Typical model code documentation includes descriptions of the model and its limitations (variable description lists, simple control flow graphs, domain diagrams, narratives, figures, equation descriptions, and references to related work) as well as information on using the code (sample input and output, and hardware and software requirements, etc.) (Trescott, Pinder, and Larson 1980; Haggerty and Reeves 2003). In order to realistically share models on a large scale, scientists have begun to define metadata standards for this information (Hill et al. 2001, Federal Geographic Data Committee 1998). CAMASE (CAMASE), SOMNET (Smith et al. 1997), and ECOBAS (Benz, Hoch, and Legovic 2001), for example, provide online meta-databases with entries covering general overview, scientific specifications, technical specifications, and contact information as well as domain-specific fields. ECOBAS provides a custom modeling language that integrates both model design and documentation into a single representation, thus eliminating inconsistencies that may arise when documentation is supplied after-the-fact without any direct tie to the model code. None of these efforts, however, include all of the information needed for coupling. UFIS (Benz, Hoch, and Legovic 2001) includes lists of variables that could be imported/exported to other specific models, but it does not indicate how or where these variables could be used in general couplings. We believe that the PCI fills this gap.

The PCI augments existing documentation and metadata with a complete description of the potential coupling behavior of a model. It is easy to construct, and to interpret. The PCI for a model code needs to be created just once; after that, any user interested in coupling can work entirely at the level of the PCI without becoming enmeshed in source code details. Finally, the PCI is directly tied to the model code removing the possibility of inconsistencies.

4 THE PCI AS A COUPLING INTERFACE: CURRENT WORK

Ultimately, PCIs will be useful only if they can serve as the basis of successful couplings. In the simplest case, the coupling of two models can be specified as a mapping of the potential coupling interface of one model to that of another as in component frameworks. Often though, more complex transformations of data and control flow are needed. In order to evaluate the use of PCIs in the fast prototyping of real applications and to develop requirements for the specification of this additional control, we have completed several case studies and present one here.

We consider a coupling of DAFlow, the surface-water flow model discussed above, with ModFlow (McDonald and Harbaugh 1988), a widely used, ground-water flow model written in Fortran.

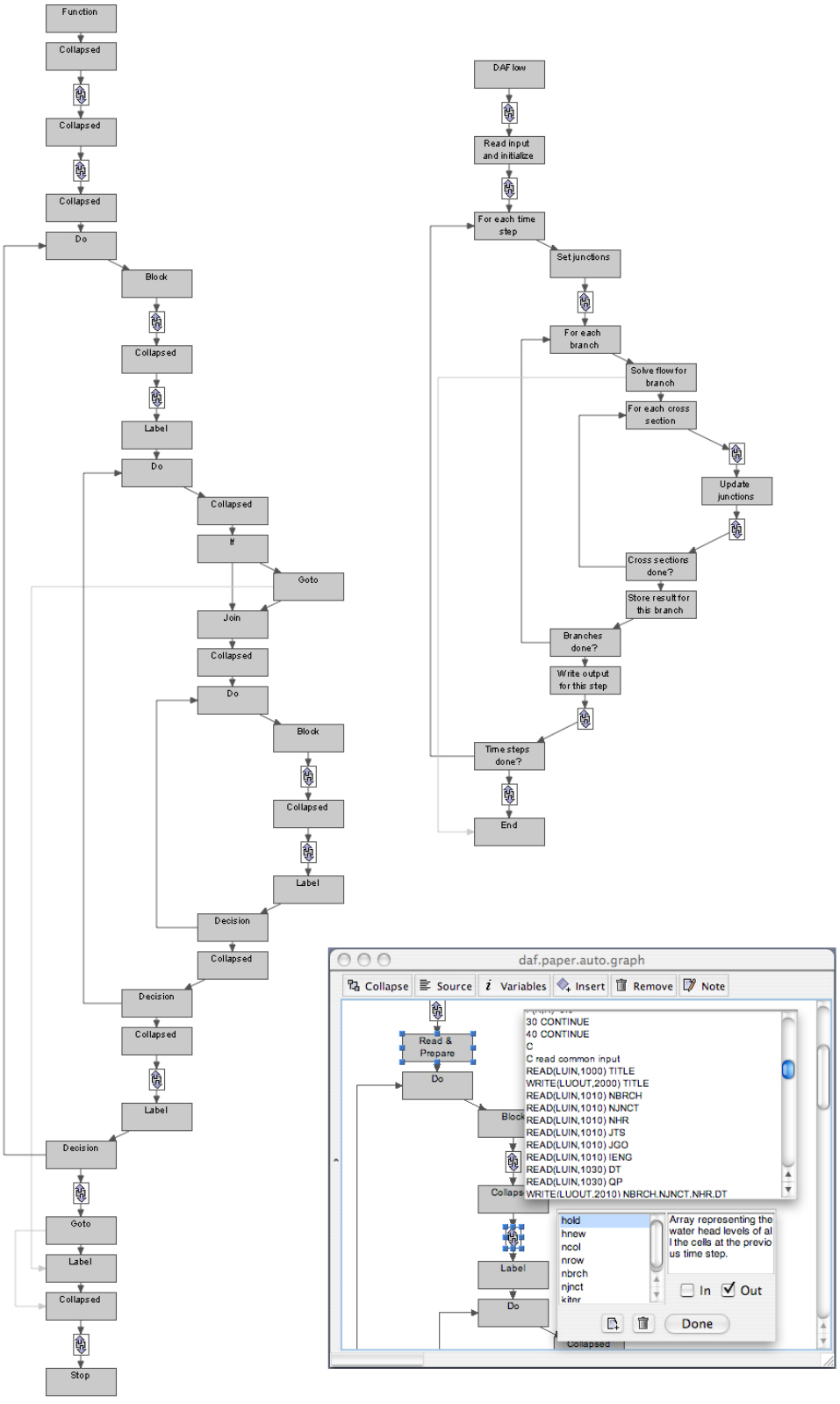

Figure 1: PCI for DAFlow: Original Reduced Graph (left), Final Graph (right), and PCI-Assist Screenshot (bottom)

The models are nontrivial, totaling about 16K lines of code. The coupling at the domain level is shown in Figure 2 where the 2d DAFlow surface network of branches is overlaid on the 3d ModFlow grid so that a set of the branch segments from the surface-water model interact with a cell along the top of the ground-water model.

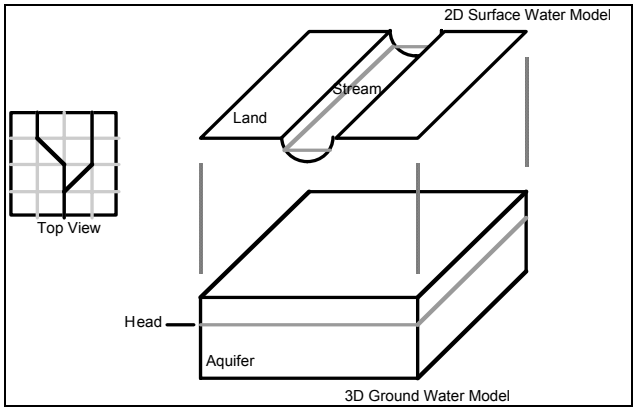

Figure 2: 2d DAFlow Surface Mapped to 3d ModFlow Volume

The purpose of this coupling is to model the interaction between ground-water flow and surface-water flow. To do so, the leakage between the two systems is calculated on each time step and used by the models in their flow calculations. If both models had the same time step size and number, then the coupling would be straightforward. But, ground water moves at a much slower rate than surface water, and thus, it is more efficient to have the ground-water model use a larger time step. As a result, DAFlow executes many time steps for each one executed by ModFlow. How should we calculate the intermediate surface-water time steps that do not have access to a corresponding new ground-water state? As is commonly done, we interpolate the needed values from the ground-water model based on its previous step, but what happens if the estimate is found to be wrong on the next ground-water calculation? The surface-water time steps calculated using the incorrect estimate must be resimulated. This requires additional control structure to be added to DAFlow so that it can repeat the simulation for a set of time steps. Since many couplings may require this type of additional control, our framework must support adding it to the PCI.

Also of concern here is the possibility that this additional control might require coupling points not specified at the time the PCI was created. We do not want to revisit the source code. In our PCI, we had included the necessary points; for the general user, we expect that the use of templates as mentioned above will help. Matching the initial CFG during PCI creation to a continuous simulation template will automatically insert the necessary points.

Figure 3 shows how this coupling might be specified using PCIs. The ellipses indicate where additional coupler code (functions executed by the coupler, using data structures stored in the coupler) is executed to perform the leakage calculations. The unshaded nodes in the DAFlow model show the added control structure discussed above. Communication between the models takes places at the double edges marked A, B, C, and D. At A, ModFlow sends initial data necessary to calculate the leakage between the ground- and surface-water, and DAFlow saves its state in the coupler, similar to a checkpoint, so that it can restore its state and repeat its calculation if the estimate turns out to be wrong. At B, ModFlow sends an estimate of its values to DAFlow. DAFlow simulates a series of time steps, and sends its result back to ModFlow at C. ModFlow then calculates its time step, and if the estimate was wrong, returns to DAFlow at B with a new estimate and the process is repeated. Otherwise, if the estimate was acceptable, both models exchange their results for the time step at D, and move on to the next time step.

This coupling demonstrates some of the additional complexities of model coupling that are not solved with the PCI. DAFlow and ModFlow have both a spatial incompatibility (grid dimensionality), and a temporal incompatibility (time step resolution). Resolving such incompatibilities is quite difficult in general, and has received a great deal of attention in the literature (MpCCI, Jobson and Harbaugh 1999, Ford et al. 2003). It is these issues though that form the intellectual task of model coupling; the PCI helps relieve the user of the more mundane aspects of coding that detract from this central task.

As a reference for evaluating our DAFlow/ModFlow coupling, we used an existing (brute-force) coupling (Jobson and Harbaugh 1999) in which the DAFlow code was divided into subroutines and integrated into the ModFlow code. As our infrastructure is not complete, we instrumented the models with communication code by hand to produce the code we expect later to produce automatically. The coupled model was then compared to the reference model in terms of accuracy, efficiency, and difficulty of coupling. In terms of accuracy, while we don't have conclusive results, the model couplings did produce the same values for all of the data we tested. In terms of efficiency, we executed both models on the same machine (600 MHz G3 iBook running MacOS X) and averaged 50 runs without screen I/O. The reference model had a mean of 3.46 seconds with a standard deviation of 0.0025 and the PCI coupled model had a mean of 9.21 seconds with a standard deviation of 0.437. Cleary there was a penalty for our prototype coupling, but we do not expect these couplings to be used for production runs. Our goal is to provide a fast prototyping environment that will enable the scientist to quickly create and test proposed couplings.

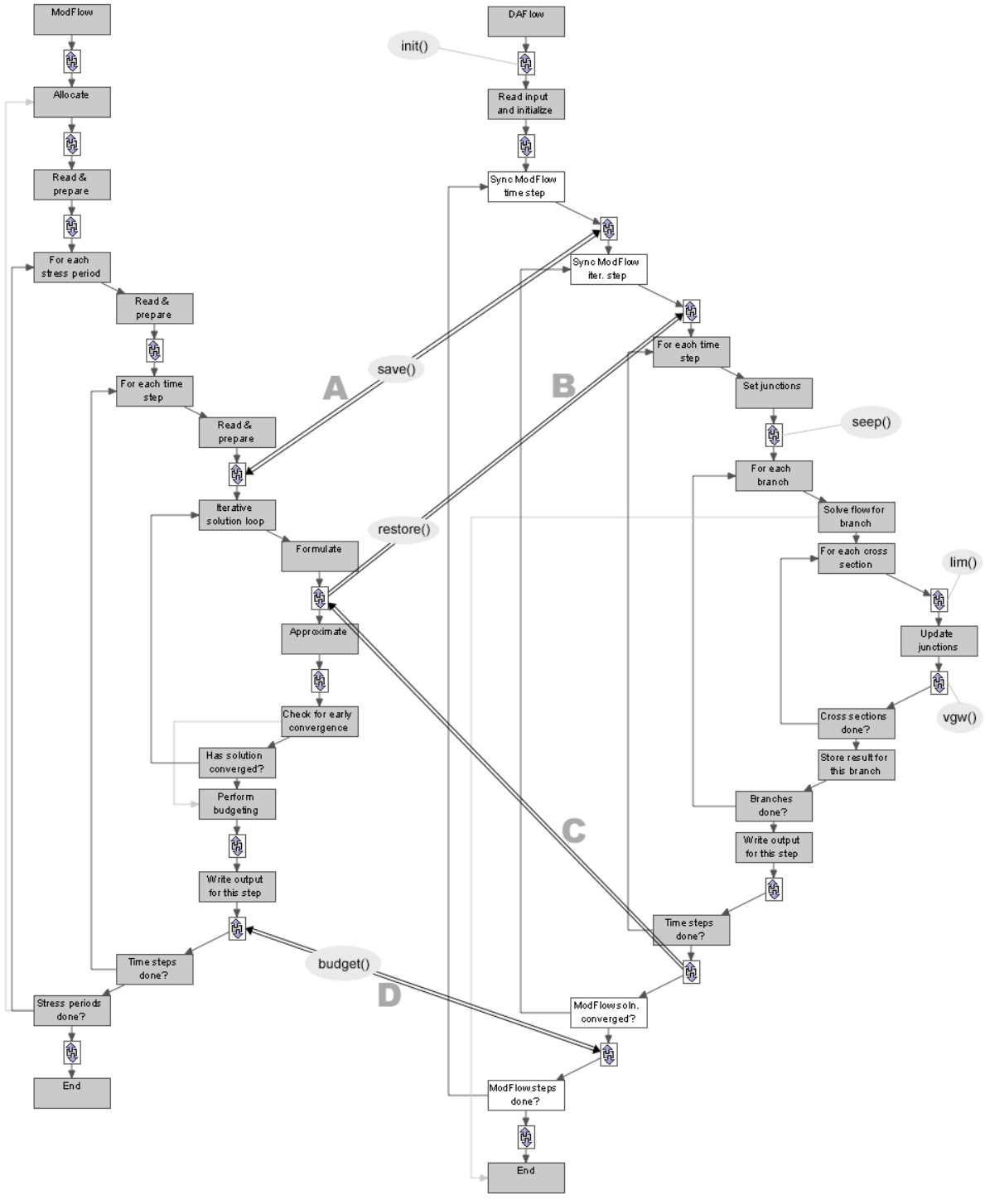

Figure 3: Complete Coupling Specification based on the ModFlow and DAFlow PCIs

The PCI couplings were created without source code modifications, allowing the user to work entirely at the higher level of the PCI, whereas the reference model was painstakingly developed through extensive recoding. In addition, our PCI need not be recreated for future couplings.

5 THE PCI AS THE BASIS FOR COUPLING CODE GENERATION: FUTURE WORK

Given linked PCIs, the final step of the coupling process is to modify the actual model source codes to reflect the coupling specification. We plan to use existing solutions in source-to-source transformations to automatically generate and instrument the model codes with the required coupling code. The direct correspondence between the PCI and the model code makes this possible. Automatic code generation is used in component-oriented frameworks but not in communication-oriented frameworks. It is important in that it relieves the user from the need to have an extensive understanding of the source code, and it prevents the introduction of additional bugs during instrumentation. Our generated coupling code will be in two forms: communication and control. The communication code will use an existing custom library to send and receive data between models and/or a coupler. The control code can introduce additional control structure into the model codes or into separate, new "coupler" components (shown as ellipses in Figure 3); it is a current focus of our research.

6 CONCLUSIONS

This paper introduces a new type of model metadata, Potential Coupling Interfaces. PCIs assist in the central tasks of model coupling: they clearly convey the code's coupling-relevant aspects; they provide a vehicle for graphically specifying the coupling of two or more codes; and they form the basis for automatic generation of coupling code. Because the PCIs provide an abstracted form of the model code, scientists can focus on the important domain and model issues of coupling without having to revisit legacy codes for each new effort. As a result, prototype couplings between a variety of models can be created quickly without an upfront investment in reprogramming. We have conducted several case studies and presented one here. Although additional research is necessary, our initial results suggest that PCIs adequately capture the coupling potential of a model and are sufficient for the specification of the coupled interaction of multiple models.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation grant ACI-0081487.

REFERENCES

Aho, A. V., & J. D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling, Volume II: Compiling*. 1st ed. Prentice Hall.

Alder, G. 2002. Design and implementation of the JGraph Swing component [online]. Available online via <www.jgraph.com/documentation.html> [accessed August 5, 2004].

Beckman, P. H., P. K. Fasel, W. F. Humphrey, & S. M. Mniszewski. 1998. Efficient coupling of parallel applications using PAWS. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing* 215-222.

Benz, J., R. Hoch, & T. Legovic. 2001. ECOBAS - modelling and documentation. *Ecological Modelling* 138: 3-15.

CAMASE. 2004. Available online via <library.wur.nl/camase/> [accessed August 5, 2004].

Federal Geographic Data Committee 1998. Content Standard for Digital Geospatial Metadata [online]. Available online via <fgdc.gov/metadata/contstan.html> [accessed August 5, 2004].

Ford, R. W., G. D. Riley, M. K. Bane, C. W. Armstrong, & T. L. Freeman. 2004. GCF: A General Coupling Framework. *Concurrency and Computation: Practice and Experience*, to appear.

Guilyardi, E., R. Budick, G. Brasseur, & G. Komen. 2003. PRISM System Specification Handbook [online]. Available online via <prism.enes.org> [accessed August 5, 2004].

Guo, W., & C. D. Langevin. 2002. User's guide to SEAWAT: A computer program for simulation of three-dimensional variable-density ground-water flow.

Haggerty, R., & P. C. Reeves. 2003. STAMMT-L: Formulation and user's manual.

Hill, L. L., S. J. Crosier, T. R. Smith, & M. Goodchild. 2001. A Content Standard for Computational Models. *D-Lib Magazine* 7 (6).

Hill, C., C. DeLuca, V. Balaji, M. Suarez, & A. DaSilva. 2004. The architecture of the Earth System Modeling Framework. *Computing in Science and Engineering* 6 (1): 18-28.

Jablonowski, D. J., J. D. Bruner, B. Bliss, & R. B. Haber. 1993. VASE: The visualization and application steering environment. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing* 560-569.

Jobson, H. E. 1989. Users manual for an open-channel streamflow model based on the diffusion analogy. U.S. Geological Survey Water Resources Investigations Report 89-4133.

Jobson, H. E., & A. W. Harbaugh. 1999. Modifications to the diffusion analogy surface-water flow model (DAFlow) for coupling to the modular finite difference ground-water flow model (ModFlow), U.S. Geological Survey Open-file Report 99-217.

Kauffman, B. G., & W. G. Large. 2004. The CCSM Coupler [online]. Available online via <www.ccsm.ucar.edu/models/ccsm3.0/cpl6/> [accessed August 5, 2004].

Larson, J. W., R. L. Jacob, I. Foster, & J. Guo. 2001. The Model Coupling Toolkit [online]. Available online via

<www-unix.mcs.anl.gov/mct/>[accessed August 5, 2004].

McDonald, M. G., & A. W. Harbaugh. 1988. A modular three-dimensional finite difference ground-water flow model. In *Techniques of Water-Resources Investigations of the United States Geological Survey,* Book 6, Chapter A1.

Mohr, B., K. Lindlan, J. Cuny, A. Malony, S. Shende, R. Rivenburgh, & C. Rasmussen. 2000. Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing* 49-59.

MpCCI 2003. MpCCI Mesh-based parallel Code Coupling Interface, Specification of MpCCI version 2.0 [online]. Available online via <www.scai.fraunhofer.de/index.php?id=222&L=1> [accessed August 5, 2004].

Parker, S. G., M. Miller, C. D. Hansen, & C. R. Johnson. 1998. An integrated problem solving environment: the SCIRun computational steering system. In *Proceedings of the 31st Hawaii International Conference on System Sciences* 7: 147-156.

Piacentini, A. 2002. PALM: A dynamic parallel coupler. In *Proceedings of High Performance Computing for Computational Science - VECPAR 2002: 5th International Conference* 479-492.

Smith, P., D. S. Powlson, J. U. Smith, and P. Falloon. 1997. SOMNET. A Global Network and Database of Soil Organic Matter Models and Long-Term Experimental Datasets. *The Globe* 38: 4-5.

Storey, M.-A. D., F. D. Fracchia, & H. A. Muller. 1997. Cognitive design elements to support the construction of a mental model during software visualization. In *Proceedings of the 5th International Workshop on Program Comprehension* 17-28.

Sydelko, P. J., K. A. Majerus, J. E. Dolph, & T. N. Taxon. 1999. A Dynamic object-oriented architecture approach to ecosystem modeling and simulation. In *Proceedings of the 1999 American Society of Photogrammerty and Remote Sensing (ASPRS) Annual Conference* 410-421.

Trescott, P. C., G. F. Pinder, & S. P. Larson. 1980. Finite-difference model for aquifer simulation in two dimensions with results of numerical experiments. In *Techniques of Water-Resources Investigations of the United States Geological Survey*, Book 7, Chapter C1.

Valcke, S., A. Caubel, D. Declat, & L. Terray. 2000. OASIS3 Ocean Atmosphere Sea Ice Soil User's Guide. Technical Report TR/CMGC/03/69, CERFACS, Toulouse, France.

AUTHOR BIOGRAPHIES

TOM BULATEWICZ, received his B.S. degree in computer science and B.A. degree in religion from the University of Rochester in 2001. He received an M.S. degree in computer science from the University of Oregon in 2003. He is currently a Ph.D. candidate in the Computer and Information Sciences Department at the University of Oregon. His research interests include distributed computing and modeling and simulation. His email address is <tomb@cs.uoregon.edu>.

JANICE CUNY, Ph. D. received a B.A. from Princeton University in 1973, an M.S. from the University of Wisconsin in 1974, and a Ph.D. from the University of Michigan in 1981. She has been on the faculty at Purdue University and the University of Massachusetts. She is currently a Professor of Computer and Information Science at the University of Oregon. Her research interests include distributed computing, programming environments, and domain-specific environments for scientific computation. Dr. Cuny serves as the Vice Chair of the Computing Research Associations Board of Directors and its Committee on the Status of Women in Computing Research (CRA-W). She also serves on the Leadership Team of the National Center for Women and Information Technology, and she is currently the 2004 Program Chair for the Grace Hopper Celebration of Women in Computing. Her email address is <cuny@cs.uoregon.edu>.

MAUREEN WARMAN, received her B.S. degree in computer science and psychology from the University of Oregon in 2001. She received an M.S. degree in computer science from the University of Oregon in 2004. She is currently working as a research assistant at the Institute for the Development of Educational Achievement at the University of Oregon. Her interests include human-computer interaction and educational technology. Her email address is <mwarman@cs.uoregon.edu>.