Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

A DOMAIN-SPECIFIC LANGUAGE FOR MODEL COUPLING

Tom Bulatewicz
Janice Cuny

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

ABSTRACT

There is an increasing need for the comprehensive simulation of complex, dynamic, physical systems. Often such simulations are built by coupling existing, component models so that their concurrent simulations affect each other. The process of model coupling is, however, a nontrivial task that is not adequately supported by existing frameworks. To provide better support, we have developed an approach to model coupling that uses high level model interfaces called Potential Coupling Interfaces. In this work, we present a visual, domain-specific language for model coupling, called the Coupling Description Language, based on these interfaces. We show that it supports the resolution of model incompatibilities and allows for the fast-prototyping of coupled models.

1 INTRODUCTION

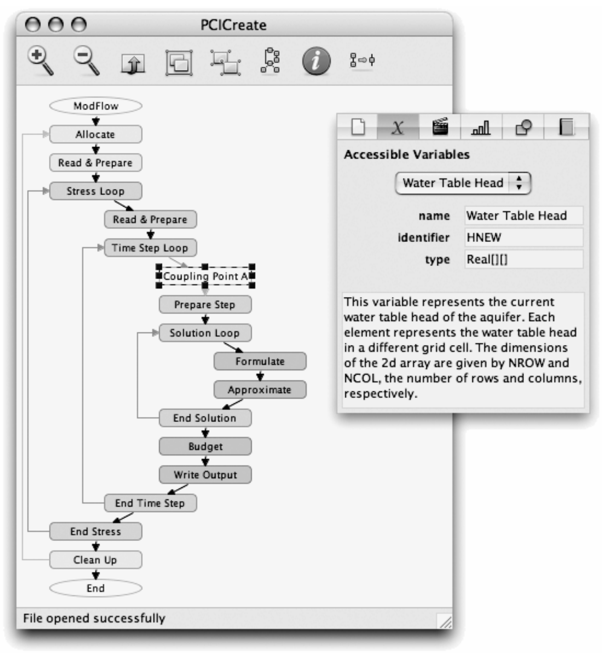
The practice of modeling and simulation within the scientific community has resulted in the creation of a wealth of validated models across many domains. The challenge now is to integrate these models into more comprehensive simulations of complex, dynamic, physical systems. Such complex simulations are often built by model coupling, that is, the merging of existing, component models so that their concurrent simulations affect each other. Model coupling is, however, a nontrivial task that is not adequately supported by existing frameworks. Current frameworks (Joppich, Kurschner, and the MpCCI team 2005; Blind et al. 2000; Ford et al. 2004; Larson, Jacob, and Ong 2005; Hill et al. 2004) often require direct manipulation of model source code, which is prohibitively difficult in many situations. We have developed an approach to model coupling that avoids this hurdle, allowing for the fast-prototyping of coupled models. InCouple, our interface-based approach, allows the scientist to work with a novel model representation called the Potential Coupling Interface (PCI) (Bulatewicz, Cuny, and Warman 2004; Bulatewicz and Cuny 2005).

The PCI is an annotated flow graph that serves four roles: first, it is a form of model metadata describing the potential ways in which a model can be used in a coupling; second it is the basis for the automatic generation of source code that is used to instrument the original model codes; third it is the vehicle for the specification of how a set of coupled models are to interact with each other; and fourth it is a reusable interface that allows couplings to be quickly prototyped. Here we focus on the third and fourth roles. We report on a novel domain-specific language for model coupling called the Coupling Description Language (CDL) that allows scientists to specify the coupling between sets of models in terms of their PCIs.

In Section 2, we provide a short introduction to the PCI. In Section 3, we present the Coupling Description Language and explain how it assists the scientist in resolving incompatibilities between models. In Section 4, we present an overview of the runtime system. In Sections 5 and 6 we demonstrate the use of the CDL with a case study of the coupling of two well-known hydrology models and present preliminary results.

2 THE POTENTIAL COUPLING INTERFACE

A PCI describes the *coupling potential* of a model, that is, those aspects of the model that in some way affect how it can be coupled to another model. They look much like the flowcharts commonly found in model documentation. The graph depicts the overall control flow of the program with respect to potential *coupling points*, that is, places where the values of state variables – the variables that represent the state of the physical quantities being modeled – can be exchanged with other models. These places are indicated by dark arrows which can be expanded to appear as blocks, such as in the example PCI in Figure 1, where the highlighted block `Coupling Point A` is expanded. Using our PCICreate software, PCIs are generated automatically from user-annotated model codes and then edited to improve readability and to incorporate additional high-level information. Block labels and colors (grayscale in the figure) in the graph, for example, are supplied by the programmer to indicate the role of the code represented by the block.



Accessible Variables

Water Table Head

name Water Table Head
identifier HNEW
type Real[][]

This variable represents the current water table head of the aquifer. Each element represents the water table head in a different grid cell. The dimensions of the 2d array are given by NROW and NCOL, the number of rows and columns, respectively.

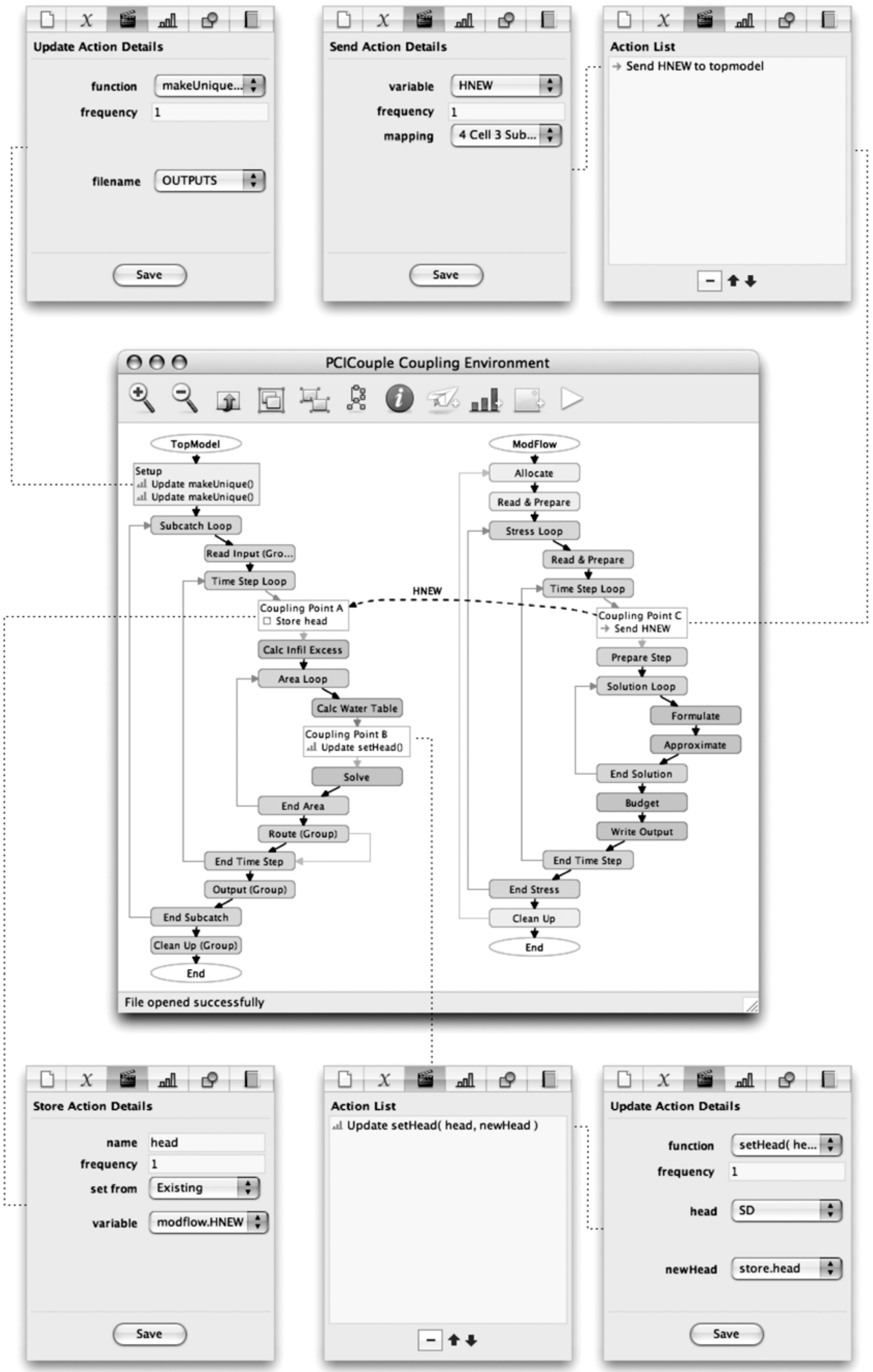
Figure 1: A PCI as It Appears in PCICreate

PCIs are meant to be created just once, by the original programmer (or someone familiar with the model), as model metadata; after that they can be reused for any couplings. The variables accessible at each coupling point (as shown in the smaller *inspector* window in the figure) are specified by the creator of the PCI. When a PCI is created, the original model source code is automatically instrumented with communication code, and after it is compiled, the resulting *coupling-ready executable* can be used in any coupling. The communication code enables the model to send and receive any variable at any coupling point, although only a subset are sent and received in a particular coupling. The primary role of the PCI is to serve as a context for describing coupled models, in which the PCI acts as a template. In the next section we present a language for describing coupled models in terms of PCIs.

3 THE COUPLING DESCRIPTION LANGUAGE

Scientists describe the interactions between models in the coupling environment (CE), provided by our PCICouple application. A coupling specification begins with the PCIs for one or more models that are to be coupled. Figure 2 shows an example of a coupling description with two PCIs. The center window shows the PCIs for the models TopModel (Beven 1997) (left) and ModFlow (McDonald and Harbaugh 1999) (right). The dashed arrow indicates that data is sent from Coupling Point C in ModFlow on the right to Coupling Point A in TopModel on the left. State variables are changed at Coupling Point B and at the `Setup` block. The environment incorporates the Coupling Description Language which consists of a set of actions that operate on the accessible variables at coupling points. To describe a coupled model is to specify a set of actions, called an action list, for each of the relevant coupling points. The action lists for Coupling Points C and B are shown in the top-right and bottom-center of the figure, respectively (there is only a single inspector window in PCICouple; several screenshots of the inspector were collected in the figure and the dotted lines were added to indicate which blocks are associated with which inspector windows). During execution of the coupled model, when a model reaches a coupling point, the actions in its action list are carried out. There are three kinds of actions: Send, Update, and Store. Collectively, they allow the values of a model's variables to be changed based on the values of other variables from that model, or from other, coupled models.

Send Action: The Send Action allows the value of a variable at one coupling point to be used at another coupling point. Send Actions are explicitly depicted in the coupling environment by a labeled, dashed line between the source and destination coupling points in the PCI templates, as shown in Figure 2. Solid lines indicate the flow of control, and dashed lines indicate the flow of data. In the figure, the value of variable `hnew` is sent from Coupling Point C in ModFlow, to Coupling Point A in TopModel. When a scientist adds a Send Action to a coupling specification, the sent variable appears in the list of accessible variables at the destination coupling point, making it usable by the actions at that point (the variable name is prefixed with the name of the sending model to avoid name conflicts). Each kind of action has a set of properties that are set by the scientist when an action is added to the action list of a coupling point. The properties of the Send Action are *variable*, *frequency*, and *mapping*, as shown in the center-top of Figure 2. The *variable* property is the name of the variable to send, which is selected from the list of accessible variables at the coupling point. The *frequency* property indicates the activation frequency of the action, which is how often the action is performed. Normally, the actions in a coupling point's action list are carried out each time execution reaches that point. In some cases though, in order to match the execution rates of different models, the interaction needs to occur at a lesser frequency. This is the purpose of the activation frequency property. It describes how often the action should be performed. For example, an activation frequency of 1 indicates that the action should take place every time the coupling point is reached, while a frequency of 4 indicates that the action should take place every 4th time the coupling point is reached. More general activation mechanisms are possible, but this simple use of frequency has been sufficient for our hydrology domain.



Update Action Details

function: makeUnique...
frequency: 1
filename: OUTPUTS

Send Action Details

variable: HNEW
frequency: 1
mapping: 4 Cell 3 Sub...

Action List

Send HNEW to topmodel

File opened successfully

Store Action Details

name: head
frequency: 1
set from: Existing
variable: modflow.HNEW

Action List

Update setHead(head, newHead)

Update Action Details

function: setHead(he...
frequency: 1
head: SD
newHead: store.head

Figure 2: The CE within PCICouple with Two Models (middle) and Various Inspector Windows (top and bottom)

The *mapping* property is the name of the data mapping that should be used for this action. It describes which individual instances should send the variable, and which should receive it. Data mappings are discussed further below.

Update Action: The Update Action is used to change the values of variables according to an update function, that is, a self-contained (stateless) subroutine that uses variables as arguments. When executed, the update function updates the value of one or more of its arguments. The coupling environment provides a collection of common functions (assign, sum, average, etc.) and the scientist can add their own. A function is written in the same programming language as the model within which it is used. The scientist chooses the update function from the list of available functions, and assigns a variable to each of its arguments, chosen from a list of accessible variables. The properties of the Update Action are *function*, *frequency*, and the argument assignments, as shown in the top-left and bottom-right of Figure 2.

Store Action: Store Actions provide a way to create *stored variables*, which are new, independent, mutable variables that do not exist in any of the model codes. They are accessible at all the coupling points in a PCI and are usable in Send and Update Actions. Each time a Store Action is activated, the stored variable associated with the action is set to either a (constant) value specified by the scientist, or to the value of a model variable, overwriting any previously stored value. The latter case allows for the value of a variable at one coupling point to be accessed later, at a different coupling point. Stored variables can be used as arguments and updated in multiple Update Actions, providing a way for the result of one Update Action to be used in another Update Action, or in a subsequent activation of the same action. Each instance of each model has its own private data memory in which these variables are stored. The properties of the Store Action are *name*, *frequency*, *set from*, and *variable*, as shown in the bottom-left of Figure 2. The *name* property is the name of the new stored variable. If its value is to be set to a model variable, then the *set from* property is set to “Existing”, and the variable is selected from a list of model variables and assigned to the *variable* property. If its value is to be specified by the scientist, then the *set from* property is set to “New” and the scientist specifies the value, data type, and shape of the new stored variable (additional fields appear).

These actions provide the building blocks for describing coupled models. We may want to couple different models together, or the same model to itself. In the latter case, many instances of the same model code can be coupled, where an instance is an executing model process. Thus, for example, we can couple appropriately parameterized instances of a model of a stream to simulate an entire watershed. In cases with multiple instances, showing a PCI for each instance would make the description cumbersome to create and difficult to understand. For this reason, we separate the description of a coupling into two parts. The interaction between the models is described visually in the CE in terms of the models’ PCIs, and the communication between the instances of those models is described textually in data mappings. In this way, only a single PCI is displayed in the CE for each model in a coupling, and it serves as a template that represents the behavior of possibly many instances of that model.

The data mappings are used to describe the communication between model instances. The CE describes the action lists that are to be performed at each coupling point, but which instances communicate with which? Which ones perform which actions? This is described by data mappings. A simple data mapping is shown in Figure 3 (bottom), with a partial coupling description (top).

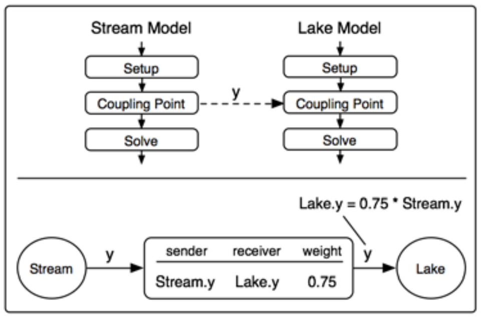


Figure 3: A Partial Coupling Description (top) and Data Mapping (bottom)

In the figure, the data mapping indicates that the value of the y variable sent from the Stream model, should be scaled to 75% before being received by the Lake model. The resulting value of the y variable is then available for use in Update and Store Actions at the destination coupling point in the Lake model, and the variable appears in the list of accessible variables at that point. Data mappings describe how one variable is transformed as it is communicated from one model to another by a Send Action. At most one variable can be sent in any single Send Action, and if two variables need to be transformed in the same way, then the specification of the data mapping can be reused.

The data mapping in the figure could represent a study site in which a stream leads into a lake. If there are several streams that lead to the same lake, each one can be simulated by a different instance of the Stream model, and coupled to the Lake model. This capability of coupling any number of instances of the same model to another model is an important feature of the CDL. The data mapping in Figure 4 indicates that there are three instances of the Stream model that communicate with a single instance of the Lake model. Since the data mapping indicates that there are three instances of the Stream model, then at least three instances of the model will be started when the coupling is executed.

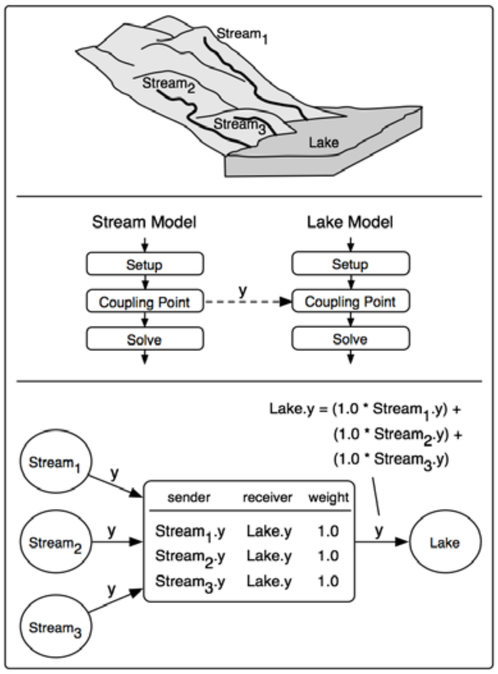


Figure 4: A Data Mapping Indicating that There are Three Instances of the Stream Model that Send to the Lake Model

In this way, each data mapping is only a partial description of the overall behavior of a coupled model. The global behavior of a coupled model is determined collectively by all the data mappings used in a coupling description. The scientist does not need to explicitly list how many instances there are in a coupling, the total number can be determined from the collection of data mappings.

Since all three instances of the Stream model are sending their value of the same y variable, those values must be combined in some way so that only a single value for y is received by the Lake model. This is because the Lake model, initially written in a different context, does not expect multiple values. The values of a variable sent from several instances of a model are combined as a weighted sum, as shown in Figure 4. Other schemes are possible, but again, this simple weighted scheme has worked for our domain.

In this example though, it is likely that each of these streams is part of a larger network of interconnected streams. Each of the streams in the network can be simulated by a different instance of the Stream model, and those instances can be coupled together. The ability to couple together instances of the same model is an important feature of the CDL. Figure 5 shows how three instances of the same model, the Stream model, can send to each other. Each instance in a data mapping is uniquely identified by a number from 1 to n, called the *instance identifier*, where n is the total number of instances in a coupled model. The data mapping in the figure indicates that instances 1 and 2 of the Stream model should send their y values to instance 3 of the Stream model. The values from instances 1 and 2 are combined, via a weighted sum, and the resulting value is received by instance 3.

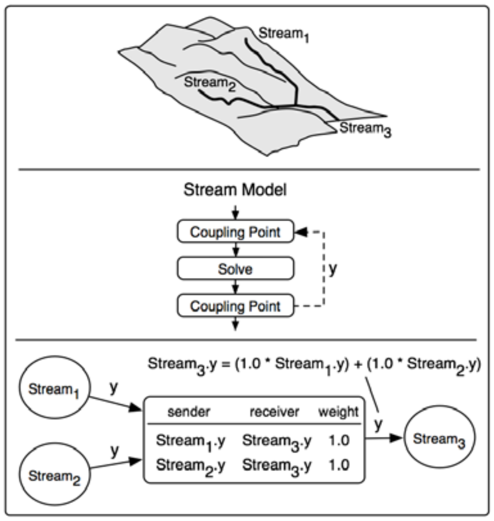

Stream$_1$
Stream$_2$
Stream$_3$
Stream Model
Coupling Point
Solve
Coupling Point
y
Stream$_3$.y = (1.0 * Stream$_1$.y) + (1.0 * Stream$_2$.y)
sender receiver weight
Stream$_1$.y Stream$_3$.y 1.0
Stream$_2$.y Stream$_3$.y 1.0

Figure 5: A Data Mapping Indicating that There are Three Instances of the Stream Model that Send to Each Other

Note that this data mapping indicates that there should be three instances of the Stream model in the coupled model. The topology of which instances send to which is determined directly from the data mapping. This topology may indicate that some instances are to receive a value, and that others are not. If an action uses a variable that is sent from another model, yet a particular instance does not receive the value, then that instance does not carry out the action. In this way, the communication topology dictates which instances carry out which actions.

For the case of sending array values, data mappings can describe how each individual element of an array is communicated and transformed between instances as well. An example of this is included in the case study in Section 5. In both cases, data mappings are text files that are formatted similar to how they appear in the figures, and are prepared by the scientist.

Send Actions are asynchronous and always execute immediately, and the sent values are queued until the receiver is ready for them. We have adopted a producer-consumer style of communication in which each Send Action results in the sending of one value, destined to a specific action at another coupling point, where it is consumed. This allows the models to remain synchronized, even if one model executes ahead of the other: the producing model can execute ahead sending values, the order of which is preserved in the receiving queue. The way in which data mappings are applied and values are queued during execution of the coupled model is described next.

4 RUNTIME SYSTEM OVERVIEW

Once the coupling description is completed, the coupled model can be executed via PCICouple. As described in Section 2, as part of the PCI-creation process a coupling-ready model executable is created for each model. The executable has the ability to send and receive any variable at any coupling point. Which variables are sent and received by each instance in a particular coupling is determined by a script that is generated from the coupling description. Each time a coupled model is executed, a set of scripts are compiled from the coupling description and one is given to each model instance. Scripts indicate which variable values are sent and received at each coupling point for each instance, and the runtime system handles the necessary transformations of these values. An overview is shown in Figure 6.

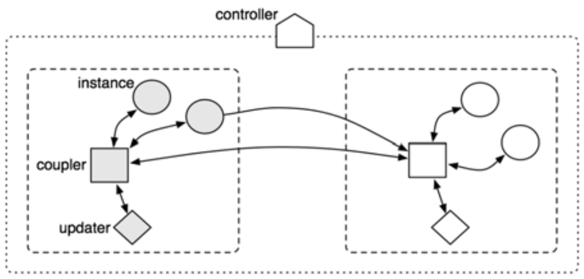


Figure 6: Overview of the Runtime System

Each shape in the figure represents an independently executing process, of which there are four kinds:

- Model Instances (circles) are independently executing model processes.
- Couplers (squares) facilitate the communication between model instances.
- Updaters (diamonds) assist couplers in the execution of update functions.
- Controllers (house shape) coordinate all the processes.

The communication between these different processes is indicated by arrows in the figure. Notice that both updaters and model instances communicate only with couplers. Every instance is assigned to a specific coupler, indicated in the figure as a dashed line, and many instances can be assigned to the same coupler. An instance's coupler provides two services to the instance, value storage and value updating:

- The coupler stores values for the instance, to support Store Actions, and acts as a queue that collects values destined to the instance, holding them until the instance is ready for them.
- To support Update Actions, the coupler applies any necessary data mappings, and then uses its updater process to carry out the execution of update functions. Each coupler has an updater process that can execute the built-in update functions and any custom functions added by the scientist.

The controller is responsible for starting all the other processes and giving them information about the coupling description. We have developed a custom communication library as part of the runtime system, based on TCP sockets, that allows models to communicate over a network. We present a case study next that ties everything together.

5 CASE STUDY

In collaboration with Alphonce Guzha, from Utah University, we have conducted a number of case studies and report on one of them here. Although the interpretation of the results is beyond the scope of this paper, we demonstrate the process of creating a coupled model using our approach.

A common use of modeling in the field of hydrology is in the study of how rainfall affects surfacewater, such as rivers and streams. One such rainfall-runoff model is TopModel, which simulates the amount of surfacewater runoff that exits an area of land in response to rainfall. It is written in Fortran and consists of approximately 400 lines of code. The amount of runoff that is generated in response to a storm event is not only dependent upon the intensity of the rainfall, but it is also dependent upon the characteristics of the land upon which it falls. The slope of the land and the type of ground cover (grass, forest, urban, etc.) are important characteristics that are taken into account in the model. Another important characteristic is the depth of the groundwater beneath the land, called the water table head. If the water table is close to the surface, then water is drawn upward and contributes to the amount of runoff, decreasing the amount of groundwater. This upward movement of water is called *baseflow*. Conversely, if the groundwater is deep below the surface, then runoff is drawn downward and contributes to the quantity of groundwater, decreasing the amount of runoff. This downward movement of water is called *recharge*. This water flux occurs through the unsaturated zone, located just beneath the land surface. In the unsaturated zone, the very small spaces between particles of dirt and sand are filled partially by air and partially by water. Below the unsaturated zone is the saturated zone, in which the small spaces between particles are filled entirely by groundwater. The upper limit of the saturated zone is the water table. An illustration of these zones and the water flux between them is shown in Figure 7.

TopModel was not designed to accurately simulate groundwater and makes simplifying assumptions regarding it: the saturated zone is in equilibrium with a steady recharge rate over an upslope contributing area, and the water table is almost parallel to the surface such that the effective hydraulic gradient is equal to the local surface slope. A more accurate simulation can be achieved by incorporating a complete simulation of the groundwater. In this study, we show how the groundwater-flow model ModFlow can be coupled to TopModel to achieve a more comprehensive simulation.

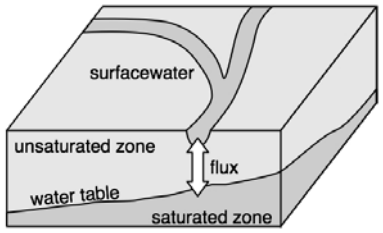


Figure 7: The Interaction between Surfacewater and Groundwater

ModFlow is a widely used groundwater-flow model developed by the U.S. Geological Society. It is written in Fortran and has approximately 10,000 lines of code. It simulates the movement of groundwater and the primary output is the water table head. In this coupling, the water table head value calculated by TopModel is replaced with the head value simulated by ModFlow. To simplify our discussion, we present only how ModFlow affects TopModel, whereas in our complete study, TopModel affects ModFlow as well.

The first step in creating the coupling is to identify which state variables in each model represent water table head. By inspection of the models' PCIs, the participating variables can be identified and are TopModel's `sd` variable and ModFlow's `hnew` variable. As described in the TopModel PCI, `sd` represents the water table depth beneath an irregularly shaped 2d area of land called a *subcatchment*. A subcatchment is an area of land that drains to a single point. In our study site (the Tenmile Watershed in Washington State), there are three subcatchments. The `sd` variable is a scalar, and represents a different subcatchment on each iteration of the subcatchment loop, visible in the TopModel PCI in Figure 2. ModFlow's `hnew` variable is an array in which each element represents the water table head in a different regular grid cell. The spatial distributions of these variables are shown in Figure 8, as they appear in the ArcMap Geographic Information System (GIS). This difference in spatial distribution of the participating variables is common in couplings and must be accounted for. In the CDL, such incompatibilities are resolved through data mappings. We first present the coupling description and then explain how the data mapping is created through the use of a GIS.

The coupling description is shown in Figure 2. As in most couplings, the models communicate within their time step loops. As long as the models start at the same point in simulation time, and use the same time step length, then the models will remain coordinated in simulation time.

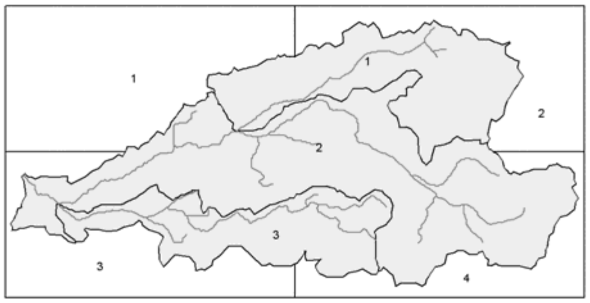


Figure 8. Three Subcatchments Superimposed on a Regular Grid with Four Cells as They Appear in ArcMap

ModFlow though, usually uses a long time step length, on the order of days or weeks due to the slow speed at which groundwater moves, and TopModel can use either a short step length to study individual storm events (on the order of hours), or a longer step length to study long term trends (on the order of days). Differing step lengths can be accommodated by adjusting the activation frequencies of actions. For example, if ModFlow uses a step length of 2 days, and TopModel uses a step length of 1 day, then the greatest frequency at which the models can communicate is every 2 days, and the activation frequencies of the actions in TopModel would be set to 2, and set to 1 in ModFlow. In this study, the models are parameterized to use a common step length of 1 day.

Inspection of the model PCIs reveals that TopModel simulates each subcatchment individually, each in a different iteration of the subcatchment loop. The time step loop is therefore executed in its entirety for each subcatchment, in our case, three times. The time step loop in ModFlow though, is executed only once (the outer Stress loop is used to coordinate the time step loop, but each time step is only executed once). Without accounting for this inconsistency, the models would remain coordinated in simulation time for the simulation of the first subcatchment only. After that, ModFlow would exit, and TopModel would begin its simulation of the next subcatchment and expect to communicate again with ModFlow. This structural incompatibility can be resolved by using three instances of TopModel, where each instance simulates a different subcatchment. Since each instance simulates only a single subcatchment, the subcatchment loop is executed only once in each instance, resulting in the time step loop being executed only once in each instance, just as in ModFlow.

The coupling description includes four action lists, one at each of the three expanded coupling points (A, B, and C), and one at TopModel's `Setup` block. We explain the purpose of each.

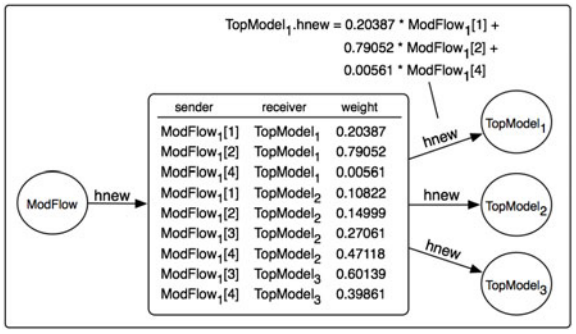
`Setup` **Block**: There are three instances of TopModel in this coupling that execute concurrently, each of which should simulate a different subcatchment. When executed though, each instance will read the same input files, resulting in all the instances simulating a subcatchment with the same characteristics. In order for each instance to simulate a different subcatchment, each instance must use a different input characteristics file. Analogous issues occur with output files as well. The variables that store the filenames used by TopModel are accessible at the `Setup` block and are the subcats and outputs variables. These variables store the filenames of the input characteristics file and the output file, respectively. Two Update Actions are added to this block, each of which applies the custom update function, makeUnique, to each of these variables, making the filename stored in the variable unique. The function simply prepends the instance identifier (accessible in all update functions via the instanceID variable) to the filenames, making them unique.

Coupling Point A: Although the value of the hnew variable sent from ModFlow needs to be used at Coupling Point B, the value is sent to Coupling Point A and stored because Coupling Point B is located within a loop, and communicating with ModFlow at that point would cause the models to become unsynchronized, similar to the subcatchment loop incompatibility discussed earlier.

Coupling Point C: Since TopModel needs to use ModFlow's hnew variable, a Send Action is added to Coupling Point C in ModFlow, which sends the variable's value to TopModel, making it accessible at Coupling Point A.

Coupling Point B: To set the value of TopModel's sd variable, an Update Action is added to Coupling Point B which applies the custom update function, setHead, which sets the value of the sd variable based on the value of ModFlow's hnew variable. Note that ModFlow's hnew value is an elevation, whereas TopModel's sd variable is a depth. In order to set the sd value to the hnew value, the elevation must be converted to a depth. This requires knowledge of the elevation of the surface, since the depth is equal to the difference between the surface elevation and the water table elevation. The surface elevation data is not present in either model, but is additional information that is incorporated into the update function and exists only in the coupled model. These custom update functions were written in Fortran and account for any differences in units between the variables upon which they operate. Here, the depth value is calculated in feet and converted into meters when the value of sd is set.

Differences in units are easy to resolve, but differences in spatial distribution are not so straightforward. Although hnew is an array representative of the water table head across a regular grid, the value received by TopModel must be a scalar that represents the water table head below the subcatchment being simulated. This transformation is accomplished via the data mapping shown in Figure 9, and is assigned to the Send Action in the coupling description. The data mapping indicates that there are three instances of TopModel and one instance of ModFlow, and it describes how the water table head values from each grid cell are weighted and combined to arrive at a value representative of the water table head below each subcatchment.



sender	receiver	weight
$ModFlow_1[1]$	$TopModel_1$	0.20387
$ModFlow_1[2]$	$TopModel_1$	0.79052
$ModFlow_1[4]$	$TopModel_1$	0.00561
$ModFlow_1[1]$	$TopModel_2$	0.10822
$ModFlow_1[2]$	$TopModel_2$	0.14999
$ModFlow_1[3]$	$TopModel_2$	0.27061
$ModFlow_1[4]$	$TopModel_2$	0.47118
$ModFlow_1[3]$	$TopModel_3$	0.60139
$ModFlow_1[4]$	$TopModel_3$	0.39861

Figure 9: Data Mapping Relating the Regular Grid to the Irregular Subcatchments

Notice that since hnew is an array, the data mapping describes how each element is communicated and transformed. For example, the value of hnew received by instance 1 of TopModel is a combination of the water table heads of the cells below it, cells 1, 2 and 4 (elements 1, 2 and 4 of the hnew array), shown in Figure 8. Specifically, the value is composed of 20.4% of the value from cell 1, 79.1% of the value from cell 2, and 0.5% of the value of cell 4.

This data mapping was created automatically via a script that we wrote in ArcMap. The grid cell polygons within ArcMap were numbered according to which element of hnew they are associated with, and the subcatchment polygons were numbered according to which instance of TopModel simulates it. The script then performs a common GIS operation called an overlay. The script determines which polygons overlap with each other, and by how much, and then generates an output file in the proper data mapping format. In this way, the precise relationship between the variables can be established. Note though, that data mappings are general-purpose and can describe the relationship between any kind of data, not just spatial data. In this case we use a GIS to create the mapping, but other third-parties could be used to create mappings between other kinds of data.

Once the model input files have been prepared, the coupling can be executed via PCICouple. The runtime system will start one instance of ModFlow and three instances of TopModel, along with any necessary couplers and updaters. The instances will communicate with each other throughout their simulations, and each instance will write its output files which can then be analyzed.

6 RESULTS

We are interested in investigating how variations in the water table head simulated by ModFlow affect the overland flow simulated by TopModel. We performed two executions of the coupled model in which ModFlow was parameterized to simulate different water table heads, and we compared the TopModel output from these runs to the output simulated by TopModel in an uncoupled simulation, which serves as a control. We created the input file sets for the models for each of the three cases using initial data that we collected for the Tenmile Watershed.

Figure 10 shows the overland flow from subcatchment 2 for a two week period during which there were several storm events.

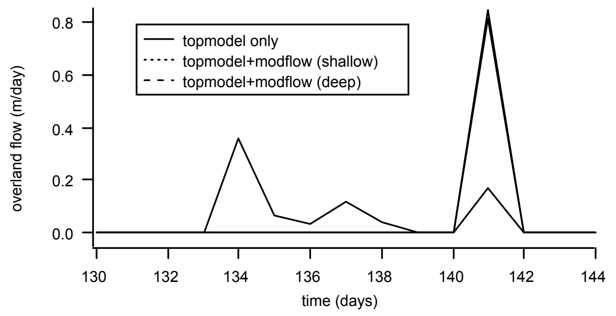


Figure 10: Comparison of the Overland Flow Simulated by TopModel in Each Case

The output of the standalone execution of TopModel is indicated by the solid line (labeled "topmodel only" in the legend), and the output from TopModel in each of the two coupled model runs is indicated by the dotted and dashed lines. The dotted line (labeled "topmodel+modflow (shallow)" in the figure) represents the output from TopModel when it was coupled to ModFlow, where ModFlow was parameterized to simulate a water table head that is half the depth as simulated by TopModel alone. The dashed line (labeled "topmodel+modflow (deep)" in the figure) represents the output from TopModel when it was coupled to ModFlow, where ModFlow was parameterized to simulate a water table head that is twice the depth as simulated by TopModel alone.

The simulated outflow in each of the coupled cases is consistent with our expectations: in the case of the shallow water table, the outflow is greater than the control case, indicating that baseflow from the saturated zone is contributing to the outflow. In the case of the deep water table, the outflow is less than the control case, indicating that the saturated zone contributes little to the outflow.

7 CONCLUSIONS

This paper introduces a new domain-specific language for model coupling. The interactions between models are described visually, in terms of the models' PCIs. Through the use of only three constructs, the actions, the way in which the variables of different models affect each other can be described. We have presented a case study that demonstrates the process of creating a coupling through our approach. We showed how the complex relationships between model variables can be resolved in couplings via data mappings, which can be automatically created. We conclude that the ability to couple models without dealing with model source code significantly reduces the time required to design a coupling to the point that coupled models can be quickly prototyped.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation grants ACI-0081487 and SBE-0318372. The authors thank Alphonce Guzha for providing domain expertise in the design of the TopModel/ModFlow coupling.

REFERENCES

Beven, K. J. 1997. Topmodel: a critique. *Hydrological Processes*, 11(9): 1069-1085.

Blind, M. W., A. Ubbels, L. R. Wentholt, Th. L. van Stijn, A. H. Bakema, J. D. Bulens, J. J. Noort, B. van Adrichem, J. Stout, and F. C. van Geer. 2000. Towards a well-oiled model infrastructure for water management: the generic framework water program. In *Proceedings of HydroInformatics 2000*. Cedar Rapids, IA.

Bulatewicz, T., J. Cuny, and M. Warman. 2004. The potential coupling interface: Metadata for model coupling. In *Proceedings of the 2004 Winter Simulation Conference,* Piscataway, New Jersey: Institute for Electrical and Electronics Engineers.

Bulatewicz, T. and J. Cuny. 2005. Interface-based support for model coupling: Spatial representation and compatibility issues. In *Proceedings of the 2005 GeoComputation Conference,* Ann Arbor, MI.

Ford, R. W., G. D. Riley, M. K. Bane, C. W. Armstrong, and T. L. Freeman. 2004. GCF: A general coupling framework. *Concurrency and Computation: Practice and Experience*, 18(2): 163-181.

Hill, C., C. DeLuca, V. Balaji, M. Suarez, and A. DaSilva. 2004. The architecture of the earth system modeling framework. *Computing in Science and Engineering*, 6(1): 18-28.

Joppich, W., M. Kurschner, and the MpCCI team. 2005. MpCCI - a tool for the simulation of coupled applications. *Concurrency and Computation: Practice and Experience*, 18(2): 183-192.

Larson, J., R. Jacob, and E. Ong. 2005. The model coupling toolkit: A new Fortran90 toolkit for building multiphysics parallel coupled models. *International*

Journal for High Performance Computing Applications, 19(3): 277-292.

McDonald, M. G., and 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.

AUTHOR BIOGRAPHIES

TOM BULATEWICZ received a B.S. and B.A. from the University of Rochester in 2001, an M.S. from the University of Oregon in 2003, and a Ph.D. from the University of Oregon in 2006. His research interests include distributed computing, domain-specific environments and modeling and simulation. His email address is <tomb@cs.uoregon.edu>.

JANICE CUNY 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. Currently she is a Professor of Computer and Information Science at the University of Oregon but is on leave at the National Science Foundation where she runs the Broadening Participation in Computing (BPC) initiative. Her research interests include distributed computing, programming environments, and domain-specific environments for scientific computation. Her email address is <cuny@cs.uoregon.edu>.