Modeling Data Product Generation

Bill Howe
bill@cse.ogi.edu

Dave Maier
maier@cse.ogi.edu

Introduction

We have been working closely with environmental scientists who develop and maintain a data product generation engine to support an Environmental Observation and Forecasting System used to study the Columbia River Estuary. Provenance issues have been encountered as artifacts of the complexity and variability of the legacy system, and we have made some observation about their nature. Below we introduce the application and argue that a notion of multiple binding times is critical in a provenance scheme designed for data product generation.

An Environmental Observation and Forecasting System

Our domain involves an Environmental Observation and Forecasting System used to study the physical characteristics of the Columbia River estuary (CORIE). CORIE is designed to disseminate physical river data to support applications such as salmon habitability studies, search and rescue operations, hazardous spill impact assessments, and hypothetical river policy scenarios. Simulated and observed data are packaged into *data products* and distributed to researchers, environmental policy makers, and the general public via the web.

The central component of the system is a simulation module, ELCIRC. ELCIRC is based on the solution of the shallow water equations for continuity, momentum, and salt and heat balances using a Eulerian-Lagrangian finite volume method. The mesh used to discretize the domain is rather extensive: 5 variables in 3 dimensions over 72 hours of simulated time amounts to 5GB of densely packed binary data generated each day. This figure does not include the data products generated from the model output. Clearly, disk and compute resources are at a premium despite falling hardware prices.

Data products are currently generated via C codes initiated via Perl scripts and Unix cron jobs. Obviously, defining and incorporating new data products is not trivial with such an architecture. To this end, we are designing a framework to ease data product specification and generation.

There is certainly a need for developers and scientists to be able to define and produce one-off products on an ad hoc basis. However, in the CORIE system, data products are also generated via *data product lines*, where the 'same' data product is generated repeatedly over a range of inputs. While the goal of a single data product is to dynamically explore a particular data set in detail, the goal of a data product line is to provide thorough coverage over some parameter space. For example, after having implemented a purportedly more accurate model, one might visualize key regions using ad hoc data products. Once the model has been proven, one might want to regenerate an entire product line, that is, re-compute individual data products for each day of the prior month using an identical grid, identical forcings (simulation boundary inputs), and identical parameters. The result is a comprehensive suite of data products that can be compared with each other meaningfully.

Documenting the Product and the Process

Provenance information associated with individual data products can provide additional context needed to make full use of the data. However, modeling the system as a collection of data products is too restrictive. Consider the following:

- You are browsing through a product line consisting of transect images over mesh version 3 for the month of July. An image exists for each day in the first half of the month, but there are none for the second half. You would like an explanation for the missing data. But how do we associate provenance information to the *lack* of a data product?
- Browsing another set of the published data products, you notice that isoline images for temperature are grouped into "hindcasts" and "forecasts." You imagine that these terms refer to the data or process used in generating the images, but specifically, what do they mean?

These scenarios illustrate the need for provenance information stored and manage at the *process* level rather than the *product* level. Both types of provenance information have to do with communicating context to data consumers, but the mechanism by which the information is stored and queried is likely to be quite different. To communicate the production process, a more general notion of provenance must be adopted.

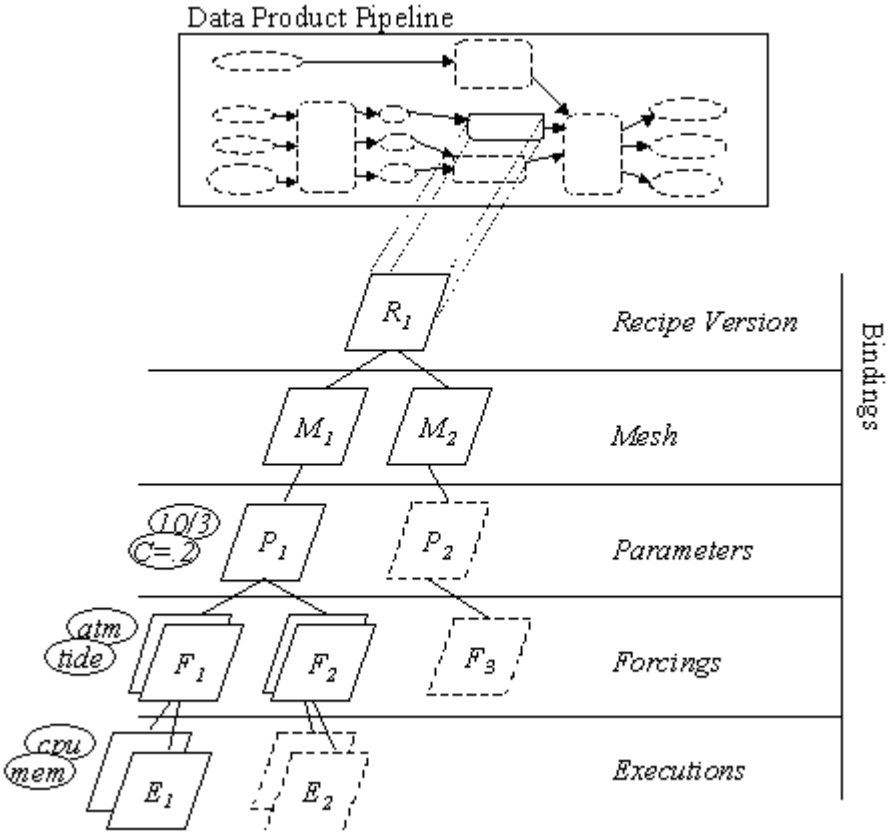

Figure 1: Illustration of Binding Levels for a single processing step

Provenance via Binding Levels

We suggest that a data product's identity is captured by a set of external objects that the product is incrementally bound to. In the absence of a precise characterization for these objects, we some examples from our domain to illustrate the idea:

code version
river bathymetry data
horizontal mesh definition
vertical layers definition
atmospheric forcings
tidal forcings
estimated river discharge (for forecasts)
observed river discharge (for hindcasts)
algorithm control parameters
initial conditions
compute environment
processor (in a parallel environment)
unix process id
memory usage measurement
running time measurement
exit status

Intuitively, a data product *line* holds some of these objects fixed while varying others. How can we flexibly capture all the degrees of freedom in which a data product can vary without needlessly duplicating objects that stay fixed? We are investigating the use of *binding levels* to model our observations that data products, and even individual processing steps, acquire their identity in stages.

We suggest that data product generation should be modeled at multiple levels representing these different stages of binding. Some systems do separately model programs, programs bound to their inputs, and program executions; these systems are applying the concept of binding times but are fixing the number and type of binding levels. We extend this idea by allowing the number and type of binding levels to vary from processing step to processing step, from product to product and even for a single product over time. As an example of the latter, consider the adaptation of a processing step to a parallel architecture. The identity of the data product does not seem to change with the adaptation, though a provenance scheme should factor in this information if it is to provide a complete description.

Figure 1 shows a possible binding time characterization of a single processing step. The box at the top represents a conventional boxes and arrows description (a recipe) for a data product pipeline: the rounded rectangles represent operators and programs, ellipses represent intermediate or final results, and arrows represent data flow. The *binding tree* below the pipeline describes an example set of bindings that might need to be represented for a particular processing step. At the recipe level, a particular version of the code is selected. The mesh level binds a particular mesh and geometry. The parameters level is a bit overloaded; control parameters for the simulation algorithms and data selection parameters for a product are both included here. Forcings are the data values over which the code is run, and executions are identical runs in different computing environment (or in the same environment at different times). The dashed boxes in the tree represent potential bindings used to implicitly define new products that differ from existing products only in some binding instance. Note, finally, that the order the levels are listed does not necessarily hold universally. In fact, flexibly binding data and parameters in the order they become available can be a powerful use of this scheme.

The different binding levels allow us to reason about the system at a partially bound stage. We describe several below:

- **Schema checking**. Once a mesh is bound to each step, we can verify that the components fit together not just with respect to data type, but with respect to the *extent* of the data as well. Figure 2 shows an example of the kind of *schema mismatch* that can be caught at this binding level. A transect product displays a vertical slice of the river based on a set of input points. Once the mesh is bound, the validity of these points can be verified using the mesh's embedded geometry.
- **Data structure.** Knowing the extent of the data also allows us to dynamically select good data structures. In addition to a binding a mesh, this step may require binding of parameters, such as number of time steps, to be effective.

- **Scheduling.** Parameters and mesh information together might allow us to algebraically manipulate the data product recipes, potentially identifying equivalent but more efficient expressions.
- **Data dependence**. Binding levels gives us a useful abstraction to determine the propagation of data through a system of processing steps. This information is useful for determining how widely erroneous data has affected a data product line.

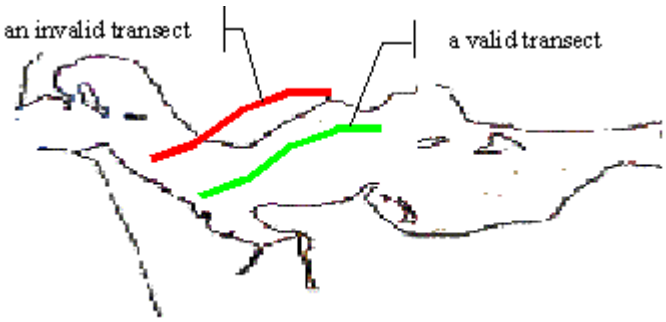

Figure 2: We can use using binding time analysis to check for invalid product recipes

Summary

Multilevel models defined according to binding times seem to be a useful abstraction for reasoning about complex data product pipelines. Binding levels allow us to generalize virtual data and discuss to the degree to which a data product is instantiated. There are several useful applications of this model, including more thorough static checking, cross-recipe optimization, and error propagation analysis.

Acknowledgements

This work is supported under NSF grant ACI 0121475 We would like to thank the entire CORIE research team for their help and insight.