Managing Soil Science Experiments Using ZOO[*] [†] (http://www.cs.wisc.edu/~ ZOO)

Yannis Ioannidis[‡] **Miron Livny** **Anastassia Ailamaki**
Arvind Ranganathan **Andrew Therber** **Maria Yuin**
Computer Sciences Department, Univ. of Wisconsin, Madison
{yannis,miron,natassa,arvind,andyt,myuin}@cs.wisc.edu

Martha Anderson **John Norman**
Soil Science Department, Univ. of Wisconsin, Madison
{anderson,norman}@bob.soils.wisc.edu

1 Introduction

Over the past three years, in collaboration with several domain scientists, we have studied the needs of a wide range of experimental disciplines, developed solutions to some of the basic problems in experiment management, and have made significant progress towards implementing a simple *Desktop Experiment Management Environment (DEME)* called *Zoo*. Our work has proceeded in a tight loop between developing *generic* experiment management technology that is implemented in a *generic* tool, Zoo, installing *customized* enhancements of the tool that constitute full systems (complete *Customized Desktop Experiment ManagementSystems (CDEMSs)*) in laboratories[1] of interest, and using the provided feedback to guide our research directions.

The overall Zoo project has been described in the 1996 VLDB Conference [7]. Specific aspects of the project and some of the Zoo modules have also been discussed elsewhere: the role of schemas in Zoo [5], the theoretical framework used for schema visualization [2] and the resulting prototype schema manager [3, 6], the data model and query language [8], and the object-to-file translator [1]. In this short paper, we first outline the overall architecture of Zoo [7] and then discuss the particular experiment that we are demonstrating at SSDBM-9.

[*] Work supported in part by the National Science Foundation ("Scientific Databases Initiative") under Grant IRI-9224741.

[†] We would like to thank all those who have been associated with the Zoo project, and especially those who have implemented significant pieces of the existing code: Ashraf Aboulnaga, Vaishnavi Anjur, Jian Bao, Shivani Gupta, Eben Haber, Anand Naryanan, Vamsi Ponnekanti, and Tom Wang.

[‡] Additionally supported in part by the National Science Foundation under Grant IRI-9157368 (PYI Award) and by grants from DEC, IBM, HP, AT&T, Oracle, and Informix.

[1] The term 'laboratory' indicates any scientific environment where experiments are conducted, be it a physical laboratory in the traditional sense, or a virtual laboratory involving scientists collaborating across the network, simulation-based modeling, etc.

Figure 1. Overall architecture of Zoo

2 Architecture of Zoo

The overall architecture of Zoo and a resulting CDEMS is shown in Figure 1. Blocks with white background are generic Zoo modules and files; for ease of reference, a short description of these modules is shown in Table 1. Blocks with gray background must be generated separately for each complete Zoo-based CDEMS. Blocks with striped background are external systems with which a given CDEMS needs to communicate.

Module	Description
EMU	Experimentation manager
FOX	Declarative object-oriented query language
FROG	Visual tool for specifying mappings between Moose objects and Ascii files
HORSE	Object-oriented database server based on Moose and Fox
MOOSE	Object-oriented data model
OPOSSUM	Visual schema manager
SQUID	Visual query manager
TURTLE	Translator between Moose objects and Ascii files

Table 1. Alphabetical list of Zoo modules with short descriptions

At the core of the system is *Horse* (*H*eavy-duty *O*bject *R*epository for *S*cientific *E*xperiments), its database server.

It is based on the *Moose* (*M*odeling *O*bjects *O*f *S*cientific *E*nvironments) object-oriented data model and the *Fox* (*F*inding *O*bjects of e*X*periments) query language [4, 8], which we have designed for Zoo. Horse is implemented using the Informix relational DBMS as a storage server, with Fox statements being translated into SQL. Moose supports various *kinds* of object classes, Any relationship from class A to class B may be specified as *derived*, implying that for each A instance, the related B instance is constructed or identified based on other objects that are (indirectly) connected to the A instance via other relationships. The construction or identification is through a Fox query and may require processing by an external system that receives as input a file containing (parts of) these other objects.

The details of the rest of the modules are not important and can be found elsewhere [7]. The role of each one will become clear in Section 4, where the operation of Zoo is described. This is done in the context of the particular experiment from soil sciences that we are demonstrating at the conference, which we describe in the next section.

3 A Soil Science Experiment

The ability to accurately model the routing of water through a watershed or agricultural area (Figure 2) is beneficial in a number of different applications. Numerous *runoff*[2] models exist to predict watershed response to rainfall, runoff, and sediment delivery, but these are typically very empirically-based and specific to the sites at which they were developed.

Motivated by this lack of general tools, we have been working to create a more physically-based runoff modeling system that will be applicable to a wide range of landscapes. The *Atmosphere-Land Exchange* (*ALEX*) model is the result of our efforts. It has been developed in the Soil Science Department at the University of Wisconsin, and has demonstrated considerable skill in reproducing surface heat and water fluxes obtained by measurements in actual land fields. Given a specification of vegetation and soil properties, and atmospheric boundary conditions at a given location within a landscape, ALEX computes each of the water flux components depicted in Figure 2 for that particular site.

Using the power of ALEX, the runoff problem can be studied with the following experiment. A grid is defined over a region of interest, and an execution of the ALEX model is performed at each cell within this grid. A complication arises because the individual ALEX executions cannot be performed independently. Water that runs off from an ALEX execution in one grid cell will necessarily run onto another cell in the grid and become an input to the

[2]Water runoff is defined as the phenomenon of water moving (running off) from one area to another (Figure 2).

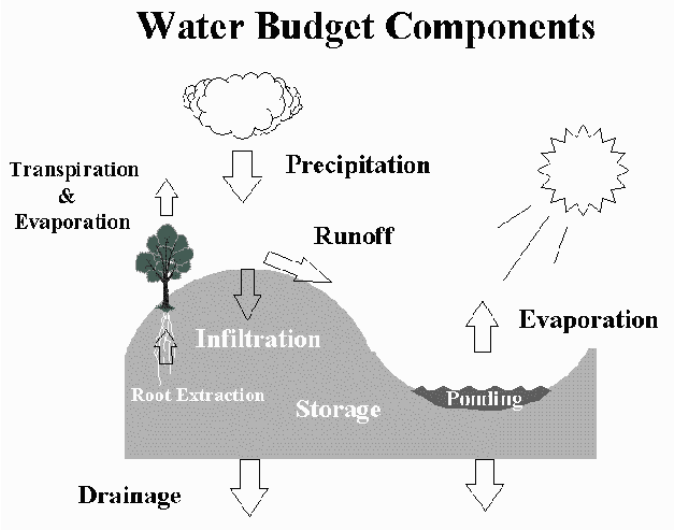

Figure 2. Hydorlogical cycling of water

ALEX execution there; the ALEX executions must therefore be able to communicate with each other. In practice, the cells are ordered and ALEX executes on them in terms of a runoff hierarchy, such that runoff from cells at higher elevations are computed before lower cells. This ordering may change as the experiment progresses; for example, the possible occurrence of ponding within the landscape can effect the order in which cells need to be processed and/or require multiple executions on the same cell (e.g., one dealing with runoff from higher cells and another, several steps later, dealing with water accumulated because of ponding).

An experiment such as this would greatly tax the computational and data management resources of the average academic workstation computing environment. The Zoo Desktop Experiment Management System, however, provides a means for realizing the experiment in a natural fashion, as shown in the next section.

4 Zoo Management of ALEX Experiments

We first discuss how the experiment may be designed using the appropriate Zoo modules and then how it is executed.

4.1 Experiment Design and Setup

The experiment designer uses Opossum (the Zoo visual schema manager) to specify the schema of the experiment, which is then used to generate a database under Horse. The structural parts of the experiment are captured as classes with interconnecting relationships, while its data and control dependencies are captured as derivation rules that are

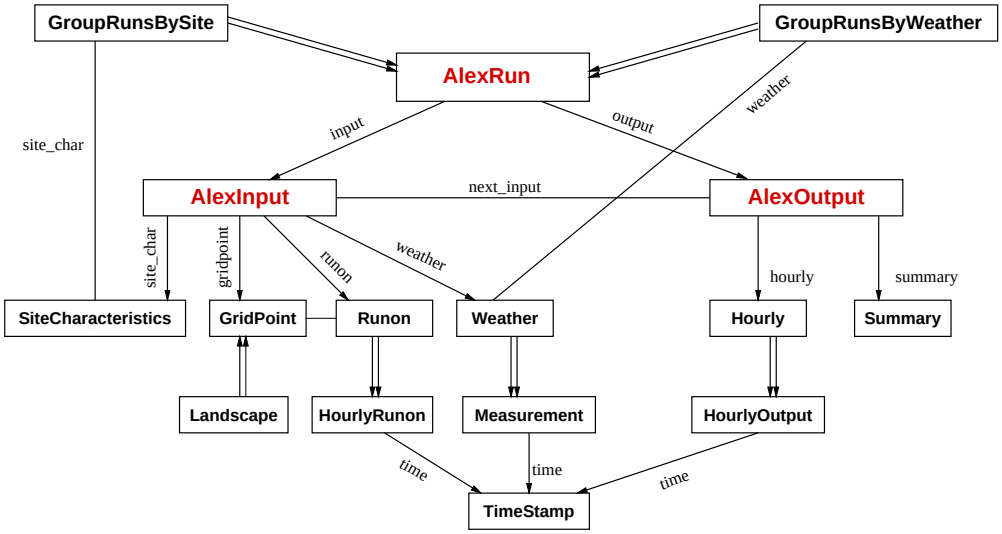

Figure 3. ALEX Experiment Schema

associated with relationships and invoke the appropriate external systems.

Part of the Moose schema for the runoff experiment is shown in Figure 3. Among its classes, three are the focal points: AlexInput describes the structure of the input needed for one ALEX execution; AlexRun captures the actual process of executing ALEX on a grid cell (specified through an AlexInput instance), and AlexOutput describes the structure of the output generated from one such execution.

The data and control dependencies in this experiment coincide, so they are all captured through the following set of derivation rules in the schema (not shown in Figure 3). (1) Associated with the 'input' relationship of AlexRun is the rule "Upon creation of an AlexInput instance, create an AlexRun instance". This rule is expressed as a Fox 'insert' command, and essentially requests the execution of ALEX on a grid cell as soon as all necessary input information about it is available. (2) Associated with the 'output' relationship of AlexRun is the rule "Upon creation of an AlexRun instance, execute the ALEX model for the corresponding AlexInput instance and store the result as an AlexOutput instance". This rule is expressed as an 'exec' Fox command, which invokes external programs, and essentially initiates the execution of ALEX. (3) Associated with the 'next-input' relationship of AlexOutput is the rule "Upon creation of an AlexOutput instance, execute the external program Next and store the result as an AlexInput instance". This rule is again expressed as an 'exec' Fox command, and essentially identifies (through the external program Next), which grid cell should be executed at any point.

In addition to designing the experiment/schema, since the experiment uses external programs (ALEX and Next), one needs to specify how Turtle (the Zoo data translator) will translate between database objects and appropriately formatted files needed by these programs. Turtle is a generic module, so it needs some customized input. The experiment designer specifies mappings between the Moose schema designed and the input and output files required by ALEX and Next. The resulting *map-files* are then stored and used as Turtle input [1]. Currently these map-files are specified "manually", using a text editor, while the implementation of a visual tool, Frog, is under way.

4.2 Experiment Execution

Having designed the experiment and the interactions of Zoo with the external systems, one is ready to execute the runoff experiment by providing the first grid cell that should be dealt with (as an instance of AlexInput). From then on the following three steps are executed for as many times as there are grid cells in the landscape:

- Create an AlexRun, pass the new AlexInput instance to ALEX and execute it.
- Receive the output from ALEX and store it as an AlexOutput instance.
- Execute Next to produce a new AlexInput instance.

This cycle is executed automatically based on the appropriate firings of the three rules mentioned above that have been defined in the schema. The experiment terminates when Next decides that there are no more grid cells to deal with, at which point it does not construct another AlexInput instance. For immediate monitoring of the experiment's status, we have provided a visual grid that indicates how many

grid cells have been computed so far (showing the order in which they were computed) and which one is currently being computed. Figure 4 shows a screendump from a 10×10 grid, at a point where 27 cells have been computed and the 28th is currently processed.

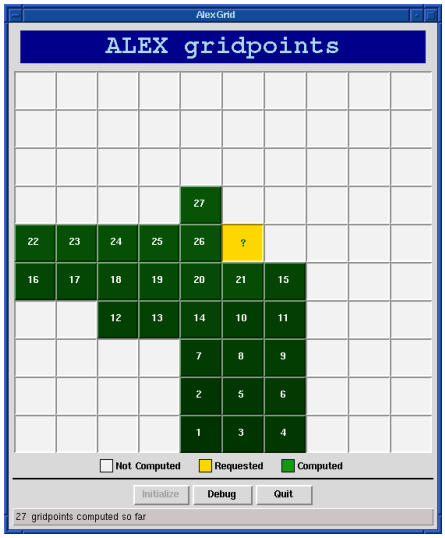

Figure 4. Screendump of grid

In what follows, we briefly describe the steps that Zoo takes to traverse the cycle above once. • After the creation of an AlexInput instance, an AlexRun instance is created via the execution of rule (1) in Horse. • This creation fires rule (2). Horse sees that this is an external program invocation and stores the name of the program (ALEX) and the AlexInput oid in an instance of a predefined class, Task. The status of this task is "unscheduled". • Emu, the experimentation management unit, polls Horse periodically for new instances in the Task class. It finds the new task, changes its status to "in-flight", and seeing that ALEX needs an input file, sends the AlexInput oid to Turtle. • Turtle, the Zoo data translator, queries Horse and retrieves all parts of the AlexInput object it received. It then uses the appropriate map-file to create an input file for ALEX. • Emu starts the execution of ALEX and receives a notification when it finishes. It then sends to Turtle the name of the ALEX output file for the reverse translation. • Turtle translates the data in the output file into objects appropriate for the AlexOutput schema and its subparts, and an AlexOutput instance insertion command is sent to Horse. • Emu changes the task status to "completed". • Horse connects the new AlexOutput instance to the AlexRun instance through the 'output' relationship. Then rule (3) is fired, and another Task instance is created for Next. • Emu finds this new task and through the same process initiates the execution of Next. Eventually, a new AlexInput is created, which starts a new cycle.

During the entire execution of the experiment (as well as afterwards, of course), scientists can use Squid, the visual query manager of Zoo, to retrieve and visualize any data that has been stored under the schema of Figure 3.

References

[1] V. Anjur, Y. Ioannidis, and M. Livny. Frog and Turtle: Visual bridges between files and object-oriented data. In *Proc. 8th International Conference on Scientific and Statistical Database Management*, pages 76–85, Stockholm, Sweden, June 1996.

[2] E. Haber, Y. Ioannidis, and M. Livny. Foundations of visual metaphors for schema display. *Journal of Intelligent Information Systems*, 3(3/4):263–298, July 1994.

[3] E. Haber, Y. Ioannidis, and M. Livny. Opossum: Desktop schema management through customizable visualization. In *Proc. 21st International VLDB Conference*, pages 527–538, Zurich, Switzerland, September 1995.

[4] Y. Ioannidis and M. Livny. MOOSE: Modeling objects in a simulation environment. In G. X. Ritter, editor, *Information Processing 89*, pages 821–826. North Holland, August 1989.

[5] Y. Ioannidis and M. Livny. Conceptual schemas: Multifaceted tools for desktop scientific experiment management. *Journal of Intelligent and Cooperative Information Systems*, 1(3):451–474, December 1992.

[6] Y. Ioannidis, M. Livny, J. Bao, and E. Haber. User-oriented visual layout at multiple granularities. In *Proc. 3rd International Workshop on Advanced Visual Interfaces*, pages 184–193, Gubbio, Italy, May 1996.

[7] Y. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. ZOO: A desktop experiment management environment. In *Proc. 22nd International VLDB Conference*, pages 274–285, Bombay, India, September 1996.

[8] J. Wiener and Y. Ioannidis. A Moose and a Fox can aid scientists with data management problems. In *Proc. 4th Intl. Workshop on Database Programming Languages*, pages 376–398, New York, NY, August 1993.