ZOO: A Desktop Experiment Management Environment* †
(http://www.cs.wisc.edu/~ZOO)

Yannis E. Ioannidis‡
Univ. of Wisconsin, Madison
yannis@cs.wisc.edu

Miron Livny
Univ. of Wisconsin, Madison
miron@cs.wisc.edu

Anastassia Ailamaki
Univ. of Wisconsin, Madison
natassa@cs.wisc.edu

Anand Narayanan
Univ. of Wisconsin, Madison
narayana@cs.wisc.edu

Andrew Therber
Univ. of Wisconsin, Madison
andyt@cs.wisc.edu

1 Introduction

Despite much interest in the area of *Scientific Database Systems* [2, 11], a major problem that many experimental scientists are still facing today is that there are no adequate experiment management tools that are powerful enough to capture the complexity of the experiments and at the same time are natural and intuitive to the non-expert. 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 Management Systems (CDEMSs)*) in laboratories[1] of interest, and using the provided feedback to guide our research directions and decisions.

The defining document of the entire project has appeared in the 1996 VLDB Conference [8]. Specific aspects of the project and some of the Zoo modules (mostly emphasizing user interfaces) have also been discussed elsewhere: the role of schemas in Zoo [6], the theoretical framework used for schema visualization [3] and the resulting prototype schema manager [4, 7], the data model and query language of the system [12], and the object-to-file translator [1]. In this short demonstration document, we first describe the overall philosophy and architecture of Zoo and then briefly discuss our experiences with the use of the current Zoo prototype. Most of this exposition is taken from the main Zoo paper mentioned above [8]. Finally, we provide an overview of how we demonstrate the system.

* 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: Vaishnavi Anjur, Jian Bao, Shivani Gupta, Eben Haber, 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.

SIGMOD '97 AZ,USA

2 Life-Cycle of Experimental Studies

We have been in an on-going dialog with experimental scientists who represent many experimental disciplines: primarily groups in soil sciences and biochemistry, but also physics, genetics, biotechnology, molecular biology, earth sciences, and manufacturing. Although these sciences have very little in common, typical experimental studies in any of them seem to go through very similar *life-cycles* [6].

We present this life-cycle through an example involving experiments in the area of soil sciences, conducted by a group of domain scientists with whom we have been collaborating the longest. They have developed the *Cupid model* [9, 10], which represents an attempt to define collective plant-environment interactions by combining knowledge from the disciplines of meteorology, soil physics, plant physiology, microbiology, entomology, and plant pathology into a single manageable package. Cupid is quite complex (more than 10K lines of Fortran) and is used in about a dozen laboratories in the U.S. and abroad. Typically about a hundred parameters are input to Cupid and over three hundred are received as output for any specific application.

Traditionally, an experimental study using Cupid goes through the following stages. *Experiment Design:* The input and output variables that are important to the study are chosen among all those dealt with by the model. This is done with pencil and paper and the final outcome is kept in notebooks. *Data Collection:* Input files are constructed in the format required by Cupid, containing the combinations of input variables to be tested. Cupid is called on each one of these files, generating each time an output file in a specific format. *Data Exploration:* Unix scripts are written to extract the required data for every different research question that the scientists may have in the course of their study.

A major impediment to exploiting the full power of Cupid has been keeping track of the numerous input and output files that are associated with a study. Over time literally thousands of files are generated, making the task of data exploration a nightmare. Another major problem has been that scientists are forced to use very different tools during each of the three life-cycle stages, making the whole process difficult to manage.

A key objective of our effort has been for Zoo to be an integrated software package with a *uniform* user interface that (a) supports the entire life-cycle of an experimental study allowing smooth transitions between its stages, (b) transparently manages all the data generated by the study, and (c) hides the details of any underlying software used. The following section describes the architecture of Zoo, which has been influenced significantly by the above objectives.

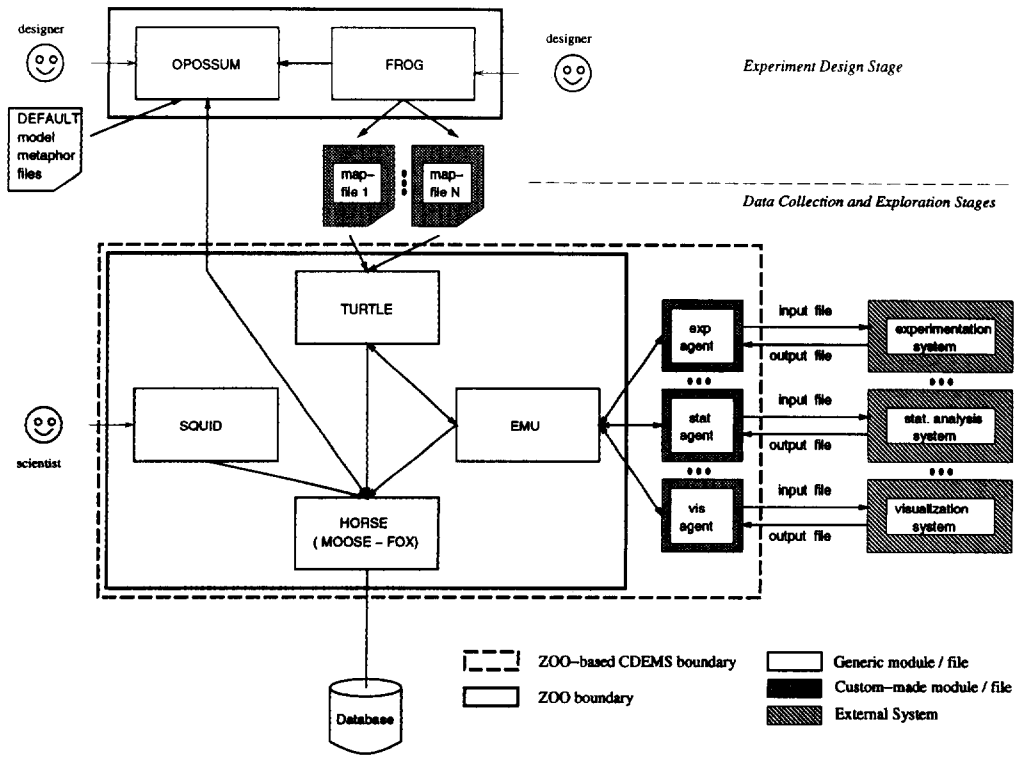

Figure 1: Overall architecture of Zoo

3 Architecture of Zoo

Zoo is designed to be a generic Desktop Experiment Management Environment (DEME). To become a complete Customized Desktop Experiment Management System (CDEMS) and be installed in a specific laboratory, e.g., the Cupid laboratory, it must be enhanced with some custom-made pieces, which can be generated usually with little effort. 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. Among them there is *at least* one experimentation system[2], where the experiments are conducted during the data collection stage. In addition, there may be other external systems that are useful in the data exploration stage, e.g., for statistical analysis or visualization.

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

3.1 Zoo Module Functionality

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 [5, 12], 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, including *tuples*, *sets*, *multi-sets* (bags), and *indexed-sets* (i.e., arrays indexed by arbitrary collections), as well as five *kinds* of binary object relationships: *has-part* (from tuples), *set-of* (from any collection), *indexed-by* (from indexed-sets), *association* (between any object kinds), and *is-a* (with the usual meaning). Any relationship from class A to class B may be specified as *derived*, implying that for each A object, the related B object is constructed or identified based on other objects that are (indirectly) connected to the A object via other relationships. The construction or identification may be through a Fox query, or may require processing by an external system that receives as input a file containing (parts of) these other objects.

Figure 2 shows a simple Moose schema in graph form. It captures a (simplified) soil-science study. Each *Experiment* is modeled as a complex object, with sub-objects representing its *Input* and its *Output*. Its output is a pair of the total *yield* and *quality* of the harvest. Its input consists of the *Weather* and a *Plant_community*, which is an indexed-set of *Plants* indexed by the set of land *Zones*. (The zone where each plant is grown is recorded independently for each Plant_community in which the plant participates.) The weather is captured by *rainfall*, *temperature*, and *wind-speed* values, and may be *windy*, in which case *wind-direction* becomes important as well,

[2] We use the term 'system' in a general sense, to include both software systems and physical systems possibly involving humans in their operation.

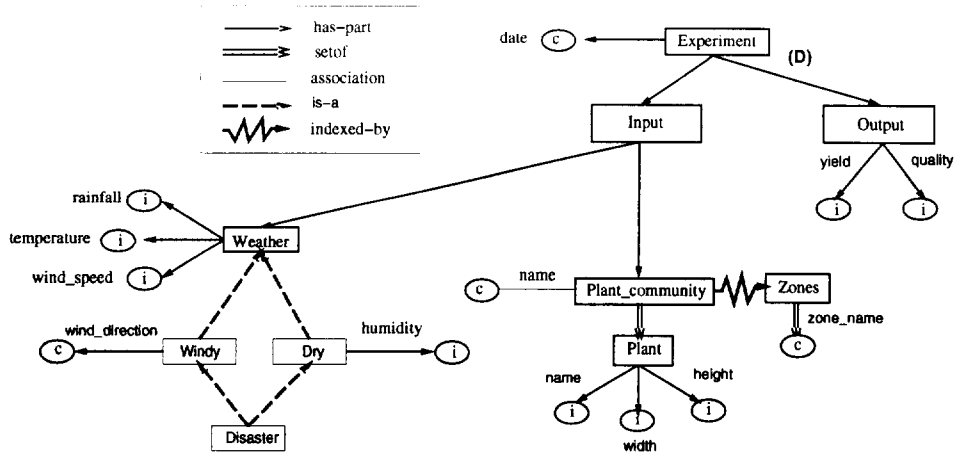

Figure 2: Sample Moose schema of Soil Sciences experiment

dry, in which case air *humidity* becomes important as well, or *disaster*, which combines the two. Note that the output of an experiment is indicated as a derived relationship (label (D)). Although not shown in the figure, the derivation is based on the input part of an experiment (in particular, the values in the primitive leaf classes of that complex object class) and is realized by the execution of an external program (e.g., a simplified form of Cupid).

Opossum (*O*btaining *P*resentations *O*f *S*emantic *S*chemas *U*sing *M*etaphors) is a schema manager [4]. It has been built following a visualization framework that we have developed, which separates the data domain from its visualization [3] for maximum flexibility. In particular, it is a *generic* visual system whose inputs are files with specifications of a data model (which is always Moose for Zoo), a *visual model*, and a *visual metaphor* that indicates the correspondence between visualizations and underlying schemas. Receiving these, Opossum is customized to operate for the specific models and visualization styles. For example, the most useful visual model for Moose schemas is that of graphs and the most useful corresponding metaphor maps graph nodes to Moose classes, graph edges to Moose relationships, etc. (Figure 2 presupposes such a metaphor.) Due to their usefulness, these models and metaphors are provided as default in Zoo.

Squid (*S*ystem for *Q*ueries *U*pdates *I*nsertions *D*eletions) is a query/update manager. It is a derivative of Opossum, using schemas as query and update templates, and therefore offers the same visualization flexibility as Opossum.

Emu (*E*xperimentation *M*anagement *U*nit) is responsible for transforming user requests into actions at external systems and preparing everything necessary for these actions. It interacts with Horse for retrieving the necessary user requests and with custom-built *agents*, one for each external system that the specific Zoo-based CDEMS needs to communicate with. It also interacts with Turtle, to which it delegates the necessary object-to-file translations.

Turtle (*T*ranslation *U*nit of *R*un *T*ime of *L*arge *E*xperiments) is the system's translator from Moose objects to Ascii files and vice versa [1]. It is also a generic module; it receives as input a *map-file* that specifies how the parts of a complex object correspond to the areas of an external file, and based on that, it performs the actual translations.

Frog (*F*iles *R*elated to *O*bjects *G*raphically) is a visual tool for generating the map-files required by Turtle [1]. In one window, it has a *sample* (input or output) file of the external system, and in another, it has the Moose schema for the experiment concerned, managed by Opossum. By highlighting a specific area in the file and clicking on the appropriate part of the schema, the designer specifies what objects correspond to what area of the file.

3.2 Installation of Zoo-based CDEMS

Each external system with which communication is desired may have specialized usage requirements that are impossible to include in a generic system. Thus, installation of a Zoo-based CDEMS in a specific laboratory or for a specific study requires that, for each external system of interest, a customized *agent* is built incorporating all the details required for interacting with and monitoring the system. For example, to execute Cupid, an agent is built that takes care of all the Cupid communication. Note that installation only requires some programming in a regular programming language to build the agents but no database expertise, which is one of the goals of our effort.

3.3 Experiment Design

During the design stage of an experiment's life-cycle, Opossum is used to specify the schema of the experiment (e.g., Figure 2), which is then used to generate a database under Horse. This schema contains derived relationships corresponding to external systems associated with the given Zoo-based CDEMS. Since Turtle is also a generic module, to be able to perform the appropriate translations between objects and files, it needs some customized input. Therefore, during experiment design, Frog is also used to specify mappings between the designed Moose schema and the input and output files required by each external system. The resulting map-files are then stored and used as Turtle input [1]. Again, no real database expertise is required for experiment design, as both Opossum and Frog are visual tools offering a high-level interface.

3.4 Data Collection and Exploration

An important feature of Zoo is that it blurs the distinction between the data collection and data exploration stages if the scientist so desires. In particular, the scientist may use Squid to request results of experiments without any knowledge of whether they have been run yet or not. When they have been run, Horse retrieves the necessary information from its database and returns it to Squid for display. When not, Horse invokes the mechanism for dealing with derived relationships (recall that the output of an experiment is derived), which eventually triggers the necessary actions at the appropriate external system.

4 Status and Experience

Zoo is being implemented in C++. Not counting any visual libraries (Tcl/Tk) or database libraries (Informix) that it uses, Zoo is currently approximately 144K lines of code. Some of the functionality described above as part of the system's design is still under development: Squid provides a visual language that captures only a subset of the expressive power of Fox; Frog and Turtle are able to deal with only a subset of constructs in the Moose model; the Emu and agent functionality is provided by the same module and so a Zoo-based CDEMS only works with a single external system at a time.

Zoo has been successfully tested by the Cupid group for experimentation. A custom-built Emu/agent combination has been implemented as a single module, for communication with the Cupid simulator. The resulting CDEMS has been used to drive test runs on Cupid. The interface offered by Opossum for experiment design has played a key role in the positive reception the system has had. Moreover, the Cupid group now uses visual schemas as their reference in thinking about the model, planning experiments, and explaining the model and experiments to other scientists, with no need to deal directly with large numbers of input and output files.

In addition to the Cupid group, Zoo has also been tested by a collaborating group in biochemistry for NMR experiments run on spectrometers, with very positive results again. To a lesser extent, Zoo has also been explored by groups in Physics, Space Sciences, and Manufacturing Engineering. This has given us the opportunity to observe the effect the system may have to a diverse range of environments.

Finally, Opossum has been used as a stand-alone schema manager for the relational and E-R data models, using various visual models and metaphors. In particular, our biochemistry collaborators have used it to design a large relational schema for the Biological Magnetic Resonance Databank (BMRB), an international repository of data on biological macromolecules derived from NMR spectroscopy. Moreover, customized (through its input files) to visualize E-R schemas as the traditional E-R diagrams, Opossum is currently used in our database courses as a database design tool.

5 Demonstration of Zoo

Our demonstration mostly focuses on the Data Collection and Exploration stages of the experiment life-cycle, i.e., the lower part of Figure 1. A small number of Zoo-based CDEMSs have been installed communicating with a variety of experimentation environments. For each one, schemas and experiments have been designed, and the map-files needed for Turtle have been constructed as well. Squid is used to invoke experiments at the appropriate experimentation environments, browse the data related to past experiments, or query it and visualize the results using available tools. Some of these experiments involve sequences of external-system invocations, thus generating a *scientific workflow* that is managed by Zoo.

To emphasize the flexibility of our system architecture and the underlying implementation, some of the experimentation environments demonstrated reside on different machines, communicating with Zoo over the network. In addition, to show the versatility and power of the various system modules, we also demonstrate additional applications that have been built on top of the Horse database server.

We believe that the Zoo demonstration is of interest to several Sigmod attendees, both as database researchers/practitioners and as experimentalists (i.e., potential Zoo users. First, from the database technology point of view, Zoo offers a rather unique approach for a database system to communicate with various heterogeneous external 'systems', which in principle do not even have to be software systems - they may be hardware instruments or even humans. While most of the heterogeneous database system efforts focus on communicating with other *database* systems, Zoo focuses primarily on nontraditional external data providers (in principle, database systems are just a special case). Second, from the experimentation point of view, Zoo is a system that intends to help scientists in their experimental studies. Several of our database colleagues (and other computer scientists) have expressed how much they would like to get rid of all their convoluted file naming schemes and data extraction script files and use a system like Zoo for their experiments in comparing algorithms, data structures, whole systems, etc.

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] J. C. French, A. K. Jones, and J. L. Pfaltz. Summary of the final report of the NSF workshop on scientific database management. *ACM-SIGMOD record*, 19(4):32–40, December 1990.

[3] 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.

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

[5] 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.

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

[7] 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.

[8] 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.

[9] J. M. Norman and G. S. Campbell. Application of a plant-environment model to problems in irrigation. In D. I. Hillel, editor, *Advances in Irrigation*, volume II, pages 155–168. Academic Press, New York, NY, 1983.

[10] J. M. Norman and G. S. Campbell. Canopy structure. In R.W. Pearcy et al., editors, *Physiological plant ecology: Field methods and instrumentation*, pages 301–325. Chapman Hall, Ltd., London, UK, 1989.

[11] A. Shoshani, F. Olken, and H. K. T. Wong. Characteristics of scientific databases. In *Proc. 10th International VLDB Conference*, pages 147–160, Singapore, August 1984.

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