GRAPHICAL USER INTERFACES FOR THE MANAGEMENT OF SCIENTIFIC EXPERIMENTS AND DATA †

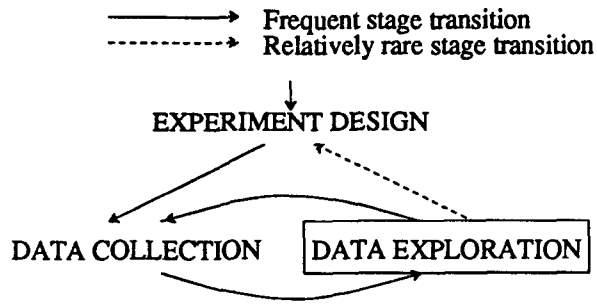
Yannis E. Ioannidis‡ Miron Livny, Eben M. Haber
Computer Sciences Department, University of Wisconsin, Madison, WI 53706
{yannis,miron,haber}@cs.wisc.edu

1. INTRODUCTION

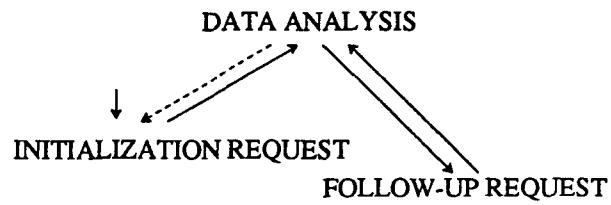
If one were to conduct a survey among researchers from all sciences asking for the greatest challenges in their particular discipline, one would expect very different answers from scientists of different backgrounds. Although this is true in general, there is at least one issue that is perceived as a major challenge in most disciplines: experiment and data management. Managing the experiment and the data produced throughout its life cycle has become the bottleneck of many experimental studies. In many cases, this significantly limits the scale of the experiments. We have reached this conclusion from our own experience with experimental computer science as well as many discussions that we have had with scientists from a wide range of experimental disciplines (Biotechnology, Genetics, Molecular Biology, Soil Sciences, Space Sciences, and High Energy Physics). While some scientists store data in hundreds of flat files or in the best case under a simple relational database system, most of them still use paper notebooks, which are clearly inadequate tools for large-scale experimentation.

In the past few years we have been involved in an effort to build an experiment and data management system that will capture the structure of data generated in experimental scientific studies. The goal of the system is to provide an integrated environment that will allow the design and execution of experiments and the access to scientific data to be done in ways that resemble the way scientists interact among themselves using pencil and paper. In this paper, we concentrate on the user interface module of the system and describe our current effort to design, develop, and test it.

The overall design of the system is motivated by a common experiment life cycle that we have identified from observations of how scientists in various disciplines conduct experimental studies. A pictorial abstraction of this life cycle is shown in Figure 1. It essentially consists of various loops traversed by the scientist multiple times in the course of a study. Some of the stage transitions occur much more often than others, and this has been captured in the figure by different types of lines.



(a) Experiment life cycle



(b) Data exploration (detail)

Figure 1: Life cycle of an experimental study.

The major stages of the life cycle (Figure 1a) are briefly described below:

- *Experiment Design:* The dependent and independent variables of the experiment as well as its environment are defined. Essentially, this defines the schema of the database in which the experiment structure and the measured information will be stored.
- *Data Collection:* Experiments are scheduled and eventually performed collecting data.
- *Data Exploration:* The collected data is studied so that conclusions can be drawn on the subject of the experiment. More details on this stage are given in Section 4.

Unlike current practice, our goal is to provide an integrated environment to scientists that will feature a uniform user interface that can be used to manage the entire life cycle of an experimental study. The data model and the user interface play the most important roles in this effort, because the success of the entire system depends to a large extent on them being intuitive to scientists, whose expertise on computer systems may be minimal.

† This research was partially supported by the University of Wisconsin Graduate School Research Foundation.

‡ The first author was also partially supported by the National Science Foundation under PYI Grant IRI-9157368 and by grants from DEC, HP, and AT&T.

In this paper, we describe the main aspects of the user interface module of the system that we are developing. We primarily focus on its features that are relevant to the first stage of the life cycle of an experimental study (experiment design). The greatest challenge posed in this stage arises from the extremely high number of parameters that must be captured. The schemas tend to be very large and the objects that they represent quite complex. Typical schemas may have several hundreds or even thousands of object classes and a correspondingly high number of semantic relationships between these classes. Thus, there are many difficulties in presenting the schema so that the scientist can grasp its overall structure while also presenting pieces of the schema that temporarily become the center of attention to the scientist. Not only are these issues important in the experiment (schema) design stage, but they also arise in later stages of the experiment life cycle, e.g., in schema-based query specification. In the next two sections, we describe our current approach to overcoming these difficulties and an example from an actual experiment in soil sciences. The subsequent section contains our plans for addressing the specific difficulties that arise in the remaining stages of the experiment life cycle with respect to the user interface.

The experiment and data management system that we are building is based on the MOOSE data model [1]. Its salient features are described below, so that the examples shown later are meaningful. MOOSE is an object-oriented data model supporting the notion of a *class* for individual objects or collection objects, i.e., sets, multisets (bags), and arrays. Every MOOSE schema has a straightforward directed graph representation whose nodes represent the object classes. Relationships between classes are captured by the graph arcs connecting the appropriate nodes. MOOSE is similar to most semantic and object-oriented data models in having two major types of arcs: *is-a* arcs (denoted by dotted lines) and *part-of* arcs (denoted by solid lines). In addition, two more types of arcs are supported that capture specialized relationships of collection objects. *Collection-of* arcs (denoted by double solid lines) connect collection classes to the classes of elements in the collections, e.g., from the class of sets of X to the class of X. *Array-index* arcs (denoted by dashed lines) connect array class nodes to the collection node(s) indexing the arrays. Other interesting features of MOOSE include two types of rules (to define *virtual attributes* and *virtual classes*) and complex user-defined structural constraints to control sharing among objects. The query language of MOOSE is very similar to SQL and has the flavor of other similar object-oriented declarative languages.

There are several other efforts at research laboratories that are similar to the one described in this paper, whose goal is to provide database support to scientific data with advanced user interfaces. These include the "Laboratory Notebook" effort in the Los Alamos National Laboratory [3] and the "Chromosome Information System" (CIS) database in LBL supported by the SDT [2] and ERDRAW [5] design tools. In both cases, when development is finished, the resulting products would correspond to major advances in helping scientists with their data management problems. The underlying data model for both efforts is the relational or the (extended) entity-relationship model.

2. THE GRAPHICAL USER INTERFACE

The user interface module of the system is implemented using InterViews 3.0 over X11. The first subsection describes the overall design of the interface and the second focuses on its experiment design tool, which is already working.

2.1. General Design

The design of the interface has been shaped by two of our previously mentioned goals for the project: (i) to provide an integrated tool to be used in all stages of the experiment life cycle, and (ii) to allow scientists to use the system in a manner that is natural to them. These have given the interface design its current form: a single graphical tool with a wide range of functionality. The interface is graphical in that the primary means of interaction between the user and the database system is manipulation of visual items through a mouse, pull down menus, and push buttons. Given that MOOSE schemas are directed graphs, the decision to develop a graphical interface was natural, since visually manipulating such graphs is clearly desirable. In addition, a graphical interface allows the use of the schema as a template for querying, so the user can see the relationships between different object classes during query specification. The range of functionality of the interface includes schema design, query specification, data entry and presentation, and manipulation of query results. By using a single tool for varied tasks, basic graphical manipulation operations, e.g., cut, paste, and move, are shared by the different parts of the system; the user only needs to learn them once, and they work uniformly whether designing a schema, querying, or viewing data.

The appearance of the user interface is not unusual: it consists of one or more windows, each with a display/work area in the middle, menus on top, tools and mode selection buttons on the left, and help messages at the bottom. Multiple windows may contain different views of the same item, e.g., different regions of the same schema, or different items altogether, e.g., a schema in one window, a query in another, and a query result in a third.

To fulfill its various functions, the interface is divided into several different modules forming a hierarchy. The general modules used by all parts of the

interface are at the bottom of the hierarchy, while the more specific tools are at the top (Figure 2). At the lowest level of the interface is the InterViews toolkit, used to drive the X-window display. The next level is the *Graphic Viewer*, which is a set of tools for displaying visual items and manipulating their appearance, e.g., move, select, copy, delete, paste, and change an item's appearance (color, size, position in front or behind, transparency-opaqueness, and alignment). Using the screen capabilities of the Graphic Viewer are the *Graph Editor*, which performs operations on graphs, and the *Data Editor*, which allows a generalized way to view data encapsulated in a visual item. Operations specific to graph editing are creation, deletion, and setting default types for nodes, arcs, and labels. The graph editor also offers several special features that make the manipulation of very large graphs easier, which are described in Section 2.2. Using the Graph Editor is the *Schema Editor*, which performs operations on the graph representations of MOOSE schemas. Specific to schema editing are the semantics for special types of nodes and arcs, testing a schema for correctness, and moving schemas to and from the database. Finally, there is the *Query Manager*, which uses most of the other modules. It provides a graphical interface for the user to specify requests to the underlying database system and produces graphical output from those requests. It consists of several subcomponents, which correspond to the various forms of querying that we envision.

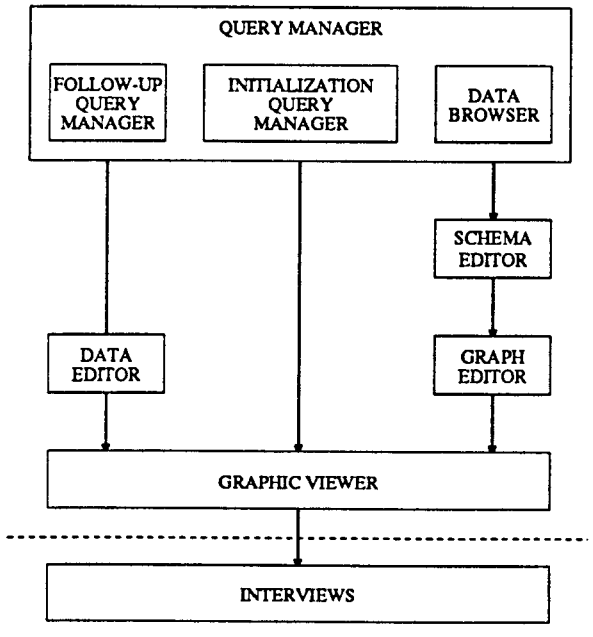


Figure 2: Architecture of the user interface.

In implementing this interface, several problems need to be solved with respect to its communication with the database system. The interface receives schemas and database objects (as query results) from the database, and sends schemas, newly entered database objects, and queries to the database. Different interface modules have different requirements from such communications, e.g., the schema editor deals with schemas only, whereas the query manager deals with schemas, queries, and data. Hence, different management strategies must be employed in the various interface modules. Since query results may contain very large collections of database objects, it is necessary to bring them to the interface in a piece-meal fashion, bringing more from the database as needed. The schema, however, is unique to each database and used by most modules of the interface, so it needs to be cached in an area that is common to all modules.

One of the most challenging tasks in this effort is identifying the appropriate internal representation of the various types of information being communicated between the database and the interface. There are two main reasons for this: (i) the needs of the different interface modules and the database are different, and (ii) the information required to display a single item may vary depending on its actual use or on its specific view shown on the screen. The schema is a good example of both (i) and (ii). In the database, it contains information not needed by the interface, e.g., pointers to data objects. In the schema editor, it contains information not needed by the query manager and vice versa, e.g., when used for database design attribute types are important, whereas when used as a template for querying attribute values are important. With respect to (ii), a schema may appear differently in multiple windows and thus require different display information in each case. The challenge with respect to (i) is whether a single representation should be chosen for the schema that will include all possible pieces of information needed by any of the components of the system or multiple ones should be devised together with a mechanism to translate among them. The challenge with respect to (ii) is to allow multiple sets of display information to be associated simultaneously with a schema in a persistent way in the database but also in a nonpersistent way for its different transient views. Currently, we are at a stage in our implementation where we are investigating the appropriate answers to these challenges.

2.2. The Schema Editor

The focus of this subsection is the schema editor, which is the only interface module that is needed for the experiment design stage of the experiment life cycle. As mentioned above, the main difficulties arise from the fact that MOOSE schemas for scientific experiments can be very large and can form an inscrutable maze of boxes and lines on the screen. Therefore, we focus on the features that have been included in the graph editor to make large schemas more manageable. These are the following: (a) allowing parts of the schema to be made invisible; (b) collapsing subgraphs into single nodes; and (c) using

"reference" nodes to eliminate very long arcs.

The simplest approach to viewing overly large schemas is to make portions of them invisible. Thus, different views of the schema can be created, with only information that is of interest to the user being visible. Selection of the parts of the graph that are made invisible can be performed in one of two ways. The first one is "manual" and requires that the user specifies each individual piece of the graph that is to disappear. The second one involves using some predefined, semantically richer operations that take advantage of characteristic properties of MOOSE schema graphs. Examples include making invisible all attributes of a selected class, the part-of subtree rooted at a selected class, or everything but the is-a arcs and the classes connected to them. The reverse operation of making unseen parts of the graph visible can only be performed using predefined operations. All operations used in the previous case can be used here as well in reverse form. Additional examples include making the whole graph visible or making visible everything that is within some distance from a selected class.

Although advantageous most of the time, the simplicity of hiding is sometimes limiting in that there are no gradations or groupings among the invisible items; items either are visible or not. Collapsing (or collectivizing) some subgraph of the schema into a single *group* node reduces the amount of visible information while imposing some structure on the invisible parts of the graph. This is very useful in encapsulating semantically meaningful parts of the schema as a single node on the screen. A group node can be named to indicate what part of the schema it replaces and always has an asterisk before and after its name (see Figure 4). All arcs connected to one of the nodes being collapsed are changed to instead be connected to the resulting group node. If more detail is required, the group may either be expanded in place or viewed and edited in a separate window. Collectivization can be nested at arbitrary levels, and the same holds for internal viewing of nested groups whether in place or in separate windows.

Another major problem that arises in large graphs representing complex schemas is that of very long arcs. For example, if many portions of the schema need to be connected with a single class, there are bound to be long arcs stretching from one side of the graph to the other. To avoid this, the schema editor provides *reference nodes*, which are denoted by circles in MOOSE schemas (see Figure 3). These are pairs of nodes that can be used to break up arcs. One of them is drawn near the source of the arc and the other near its destination. The original arc is replaced by appropriate connections of the reference nodes with the regular nodes that are respectively close to them. By using reference nodes, any schema graph can be made planar. As a way of recapturing the original arc, one can select a reference node and the system instantly moves the view in the screen to the other member of the pair. Reference nodes can be named to indicate what the remote destination of the original arc is. Thus, they act as surrogates for a distant region of the schema: an arc connected to a distant class can be attached to a local reference node instead.

3. A CASE STUDY

In this section, we report on a case study where we exposed the schema editor described above to a "real" user. Being firm believers in the value of user feedback for interactive systems, right from the beginning of our effort we tried to create a user community of researchers who are not computer scientists and who would use the graphical interface being developed. The benefits of such interactions would flow in both directions: on the one hand, the robustness and functionality of the system would be tested and our understanding of the applications needs would improve, so that the appropriate corrections and enhancements could be made; on the other hand, scientists would be able to take advantage of the capabilities of the system to improve on their established approach to experimentation. Prof. John Norman from the Soil Sciences Department at the University of Wisconsin is one member of the group of users that has been formed. The main emphasis of his work is on simulating the growth of plants based on various environmental, soil, and ecological parameters. His primary tool in his studies is the *Cupid* model [4], which simulates the necessary plant growth processes. It has been under development for about 15 years. The program that implements the model is approximately 10,000 lines of Fortran code and is being used by about a dozen laboratories in the U.S. and abroad. In the past few months, Prof. Norman has been using the schema editor that we have developed to document the structure of the input and output parameters of his model in the form of a MOOSE schema. The background and experiences from this collaboration are described in the rest of this section.

The Cupid model represents an attempt to combine knowledge from the disciplines of meteorology, soil physics, and plant physiology into a single manageable package. It defines collective plant-environment interactions at the same level of detail for the entire system as the respective disciplines have accomplished for selected parts of the system. Thus, it provides a vehicle for combining information from several disciplines to address practical problems that no single discipline can accommodate. It has been used in numerous applications that range from interpreting remotely sensed signatures of vegetation from ground, aircraft, and satellite measurements to predicting the influence of environmental factors on productivity of agricultural and ecological systems.

The basic approach in this model is to parameterize relevant processes at the spatial scale of a few

centimeters (leaf scale) and temporal scale of minutes to an hour, and then integrate to larger spatial scales of hundreds of meters (community of plants) and time scales of weeks or months (seasonal scale). For example, measurements are taken on representative leaves to obtain values for parameters that are part of the input to the model. From them the model can predict the response of a plant community or crop for a season; these responses usually cannot be measured directly.

A common belief about scientific databases is that their schema is relatively small and simple and that the challenge lies in the magnitude and complexity of the data. Cupid is one of many examples that we have encountered that invalidates the above. It simulates numerous processes that are parameterized, and therefore a large number of parameters need to be specified to characterize its input and output. Some example parameters are soil and plant conductive properties for heat and water, radiative and convective properties of leaves, the dependence of photosynthetic, respiratory, and transpiration rates of leaves on environmental conditions, and various soil and atmospheric boundary conditions. Typically about a hundred parameters are input to the model for any specific application, whereas the output variables number in the several hundreds.

By doing simulations under conditions that are not normally encountered during field measurements, the real benefit of models like Cupid is realized because insights from the studied processes are not otherwise available. However, with a hundred input parameters and several hundred output variables, testing quickly produces a mass of information that is extremely difficult to manage without some organizational tools. Computing cycles are not as limiting to progress as an efficient tool for coordinating and organizing the hundreds of files that contain input data or output results.

Our collaboration with Prof. Norman has showed that the schema editor that we have developed can serve as that tool. We have completed a schema for the input part of the Cupid model, which is shown in Figure 3. It contains more than 100 object classes. Although the schema is in reality one connected component, reference nodes are used heavily to make the picture manageable, so its visual appearance has many components. It turns out that the specific choice of connected components is not arbitrary, but follows the higher level semantics of the

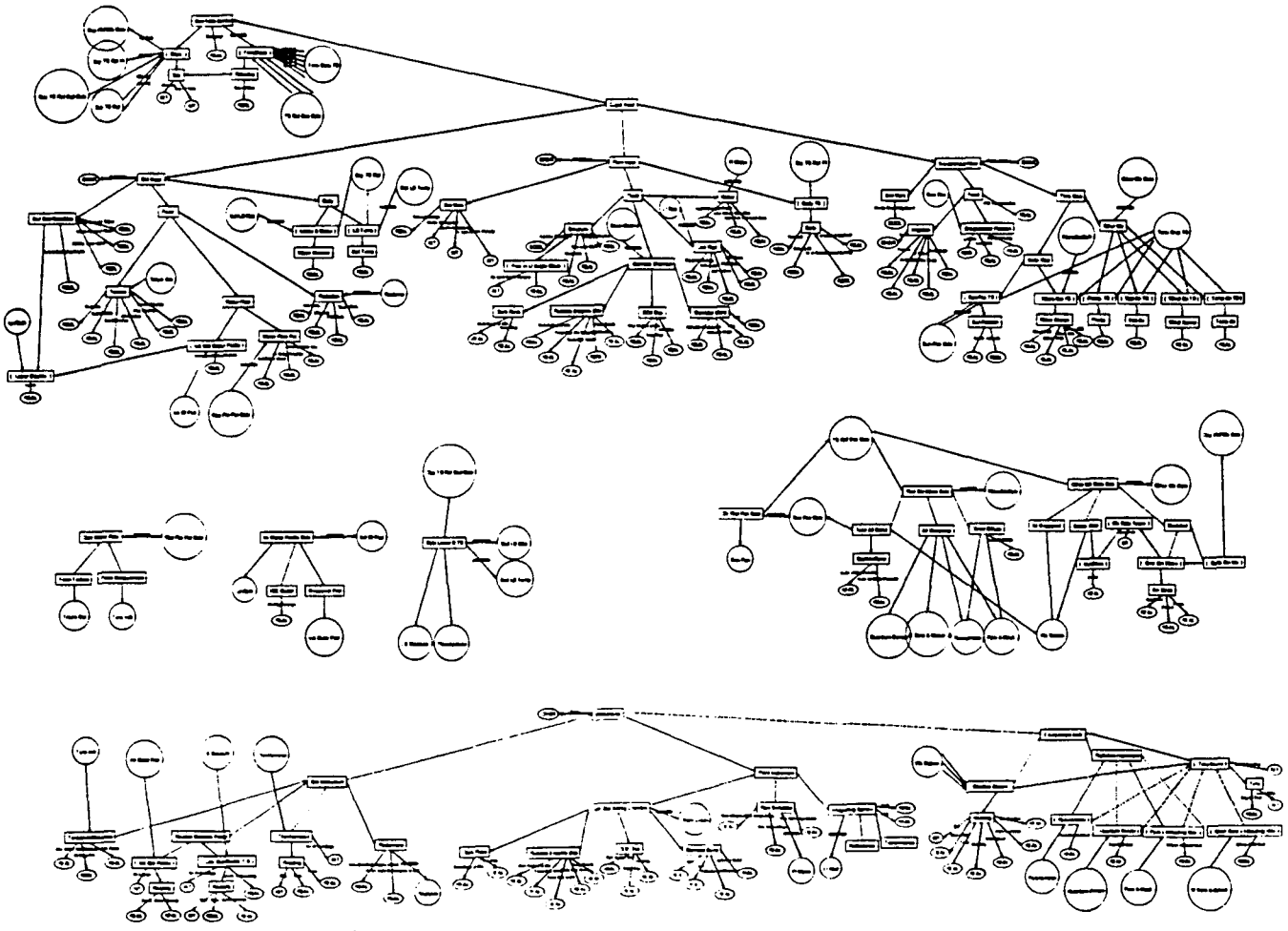


Figure 3: MOOSE schema for the Cupid input.

object classes in each component. To explain this better, we have collectivized most of the nodes in that schema into group nodes to obtain Figure 4. As one can see, there are three main categories of object classes in the schema, which essentially determine the components in the graph: those of direct input to Cupid, those of auxiliary calculations, and those of measurement instruments.

The top (part-of) subtree in Figure 4 shows the Cupid input together with groups for the three main subtrees that comprise it. The three groups correspond to the parameters that characterize the soil, the plant, and the environment, respectively. Together they include all the object classes that capture the structure of parameters that are required by the Cupid model. In a real experiment, the object instances that populate these classes are generated in one of three ways: (a) they can be directly inserted by the user; (b) they can be the output of a calculation, that is another model that simulates some aspect of the Cupid input, e.g., a model of the sun calculating radiation levels on the earth surface; and (c) they can be generated by a measurement instrument. The middle two unconnected nodes in Figure 4 are groups of object classes of two types of calculations, those modeling the soil and those modeling the environment. These have their own inputs, while their output is connected to the appropriate input object classes of Cupid (using the mechanism of derived attributes). Finally, the bottom (is-a) subtree in Figure 4 shows the instruments that can be used for direct measurements together with groups for the three main types of instruments (those for soil, plant, and instrument again). For each different class of instruments, the parameters that it can measure are connected to the appropriate input object classes of Cupid. Thus,

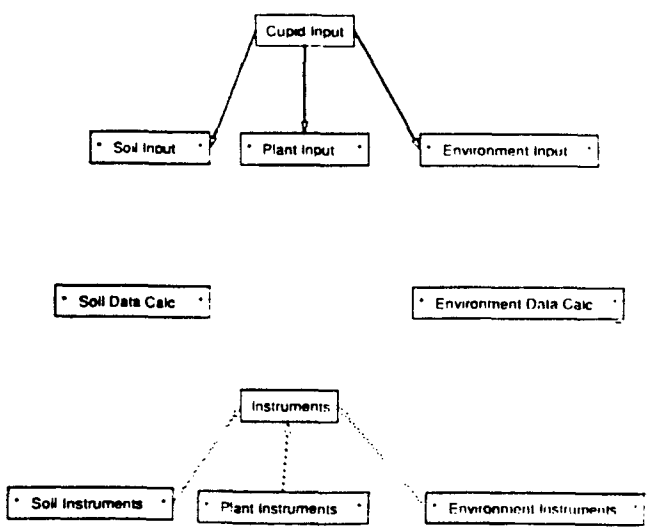


Figure 4: Condensed version of the MOOSE schema for the Cupid input.

not only does the schema capture the structure of the input for Cupid, but it also documents the relationship between the Cupid model and the other auxiliary models and measurement instruments.

Our work with the Cupid model has been yet another example for the valuable feedback that an early exposure of an interactive system to "real" users provides. A wide range of bugs of the schema editor were revealed and various limitations of its early versions were demonstrated. The whole exercise has provided us with valuable experience about the needs of scientific experiments and about the expectations of researchers that are not computer scientists. The main challenge posed by the Cupid model has been the large size of the schema. With such a schema, the user must focus on a small part of it at any given time. Realizing the above in the context of Cupid has essentially determined the emphasis of our work until now and has produced the techniques mentioned in Section 2, i.e., invisible parts of the schema, node collectivization, and reference nodes.

In the other direction, Prof. Norman and his group have benefited from the use of the schema editor as well. We let him express this in his own terms. "The main benefit of the MOOSE schema of Cupid is a clear documentation of the input structure of the model. This is the beginning of an orderly process for improving Cupid in at least four ways: (1) The eventual storage of the input data to Cupid in a database with the above structural organization will help tremendously in organizing the complex array of data combinations that are necessary for conducting simulations. (2) The creation of an object-oriented structure on Cupid data will permit other scientists to understand, use in their research, and possibly enhance with their own routines, portions of the model ignoring the rest of it. Such an exchange, which is now impossible, is essential if we are to simulate more complex systems. A clear delineation of objects could have a profound impact on the entire field of environmental biophysics, which depends heavily on models that are continually being rewritten. (3) The graphical representation of the MOOSE schema serves as a useful piece of documentation for scientists who are using Cupid in their research. (4) As a scientist, using a tool like the schema editor is helping me to understand what objects exist in the context of a soil-plant-atmosphere model."

4. LATER STAGES OF THE EXPERIMENT LIFE CYCLE

In this section, we first complete the picture of the experiment life cycle that we started in Section 1 by having a more detailed look at the data exploration stage and then briefly describe our future plans for the user interface. As shown in Figure 1b, a scientist is involved with three types of operations in this stage: (i) posing *initialization requests*, which is a time consuming process since

the values of most parameters of the experiment must be specified together with the information to be retrieved; (ii) *data analysis*, where the collected data is further processed to obtain meaningful summaries of it; and (iii) posing *follow-up requests*, which are very similar to earlier requests possibly using the answers of the latter as a reference point. This type of request represents the most common form of interaction in the course of a study.

One of the greatest challenges posed in the data collection and exploration stages arises from the complexity of the underlying objects. In both stages, complex objects must be specified to the system either as part of ordering a specific experiment or as part of posing a request to the system (selection). This will be facilitated by the use of the schema as a template for object specification but is still quite hard since it must be done graphically for a variety of object types, e.g., sets, charts, other graphical renditions of scientific data, tables, and graphs. The difficulty arises from the fact that each type of object may have multiple natural graphical representations, many of which may be different from other object types. One of our major goals is to identify a core set of representations that capture a wide variety of object types and to provide the means for specifying user-defined specialized representations for individual object types. Another goal is to implement mechanisms for explicitly or implicitly choosing among the various available representations for an object.

A second major challenge posed in the above two stages is the complexity of the relationships among objects that are represented by paths in the schema graph. The size of the schema makes the specification of these relationships by path demarcation a very time-consuming process. Allowing the specification of incomplete paths (e.g., specifying only the two end-points) which are completed internally based on the schema structure is one of our goals. The major problem that arises is resolving the ambiguities that arise when multiple plausible path completions exist. We are currently studying several mechanisms that could allow automatic disambiguation in many cases.

A third major challenge is specifically related to posing follow-up requests. The scientist should be able to use the results of previous requests as the means to make further ones. The system must identify the request that generated the previous results, modify it to reflect the specified changes, and then use it in its modified form to generate the requested data. In most cases, the difference between the two queries will be in the value of very few parameters. Maintaining the context of previous requests and interpreting the new one based on it is our basic approach to the problem. As before, ambiguities may arise from these requests as well, so several disambiguation mechanisms are under investigation.

5. CONCLUSIONS

We have described the overall structure of the user interface module of the database system that we are developing to provide support for the management of scientific experiments and data. Focusing on the schema editor of the interface, we have described its most important features, which specifically address the issue of very large schemas and how they can become more manageable. We have also presented a case study where soil scientists have used the schema editor to capture the input structure of their experiment with great success. What we have discussed in this paper is only the beginning. Several aspects of the interface are currently under investigation, primarily with respect to querying and the interface-database communication. As before, this current effort is driven by the goal of minimizing knowledge and effort required by the scientist to use the interface. The final outcome of this project, in the form of a working system, will be made available again to several scientists, who will be the final judges on the practical value of our work.

Acknowledgements: We would like to thank Prof. John Norman and his group for their willingness to use the schema editor that we have developed and provide us with valuable feedback on its problems and features. In particular, we would like to thank Larry Murdock for his patience with all the core dumps that he experienced and all the bugs that he discovered.

6. REFERENCES

[1] Y. E. Ioannidis and M. Livny, "MOOSE: Modeling Objects in a Simulation Environment", in *Information Processing 89*, edited by G. X. Ritter, North Holland, Amsterdam, The Netherlands, August 1989, pp. 821-826.

[2] V. M. Markowitz and W. Fang, *"SDT - A Database Schema Design and Translation Tool"*, Technical Report LBL-27843, LBL, Berkeley, CA, May 1990.

[3] D. Nelson, *"The Laboratory Notebook Technical Manual"*, Technical Report LA-UR 88-1256, Los Alamos National Laboratory, Los Alamos, NM, March 1990.

[4] J. M. Norman and G. S. Campbell, "Application of a Plant-Environment Model to Problems in Irrigation", in *Advances in Irrigation* II, edited by D. I. Hillel, Academic Press, New York, NY, 1983, pp. 155-188.

[5] E. Szeto and V. M. Markowitz, *"ERDRAW - A Graphical Schema Specification Tool"*, Technical Report LBL-PUB-3084, LBL, Berkeley, CA, October 1990.