CONCEPTUAL SCHEMAS: MULTI-FACETED TOOLS FOR DESKTOP SCIENTIFIC EXPERIMENT MANAGEMENT

YANNIS E. IOANNIDIS*
MIRON LIVNY
Computer Sciences Department, University of Wisconsin
Madison, WI 53706, USA

Received September 1, 1992
Revised November 1, 1992

ABSTRACT

In this paper, we identify some of the fundamental issues that must be addressed in designing a desktop *Experiment Management System (EMS)*. We develop an abstraction of the set of activities performed by scientists throughout the course of an experimental study, and based on that abstraction we propose an EMS architecture that can support all such activities. The proposed EMS architecture is centered around the extensive use of conceptual schemas, which express the structure of information in experimental studies. Schemas are called to play new roles that are not usually found in traditional database systems. We provide a detailed exposition of these new roles and describe certain characteristics that the data model of the EMS must have in order for schemas expressed in it to successfully play these roles. Finally, we present the specifics of our own effort to develop an EMS, focusing on the main features of the data model of the system, which we have developed based on the needs of experiment management.

Keywords: Scientific databases, experiment management, conceptual modeling, graphical user interfaces, object-oriented data models

1. Introduction

In the past few years, several scientific communities have initiated very ambitious and broad-ranged projects whose goals are to significantly advance the frontiers of knowledge in their disciplines by solving very hard problems that until recently were considered unapproachable. Such efforts are expected to last for many years and will play the role of umbrella projects under which several scientific questions will be investigated. The NASA Eos project and the NIH Human Genome project are two examples of national and international scientific endeavors that belong to this category. The goal of Eos is to collect data about the earth and its atmosphere that will be used by earth scientists for global-change research, while the goal of the Human Genome project is to sequence the human DNA and from that understand the

*Partially supported by the National Science Foundation under Grant IRI-9157368 (PYI Award) and by grants from DEC, HP, and AT&T.

nature of genetic diseases. In this paper, we use the term *global project* to refer to a large-scale scientific effort like the ones above. A major component of such projects is the collection of measurements on complex phenomena. Such activities will generate huge amounts of data (sometimes measured in petabytes—one petabyte is equal to $10^1$5 bytes), which will then be studied by thousands of researchers. Managing this surge of scientific data poses many challenges, with which current database technology is unable to deal. Several technical problems need to be solved before *Scientific Database Systems* can become a reality. An excellent account of these problems together with an overall picture of the major scientific projects that are currently under way is given in the summary of the NSF Workshop on Scientific Database Management [FJP90].

The widespread availability of the unprecedented collections of data gathered as part of the above projects will generate much scientific activity at the level of individual scientists or small teams of scientists. Smaller projects will be initiated to study a variety of phenomena related to the global projects, using small fractions of the available data. In this paper, we use the term *local study* to refer to such smaller-scale research efforts[1]. Given the scale of such studies, it is desirable that the experiments and the data generated from them be managed directly by the scientists themselves, who will not be experts in database systems. There are no adequate management tools, however, that are natural and intuitive to the non-expert and offer the desired functionality. Thus, similarly to the large-scale projects, these smaller studies will also suffer from the lack of appropriate technical support.

The above is perceived as a major problem for experimental studies in most scientific disciplines even today. Based on our own experience with experimental computer science [Liv87] and from joint work that we have undertaken with scientists from a wide range of experimental disciplines (biotechnology, genetics, earth and space sciences, soil sciences, and high-energy physics), experiment and data management have become the bottleneck in such studies. In many cases, the lack of adequate management solutions significantly limits the scale and scope of the experiments. 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 extensive experimentation.

There are some technical challenges that are unique to each of the two types of activities mentioned above, i.e., managing the collection and distribution of the primary data for a global project and managing a local experimental study (which may or may not use data collected within a global project). For example, dealing with large amounts of data is primarily an issue in global projects. On the other hand, supporting scientists who are not experts in databases so that they manage the execution of experiments themselves is only an issue in local studies. Nevertheless, many problems are common to both types of activities. Examples include the types of data, the size and complexity of the structure (schema) of the data collection process and/or experiments, and the need to provide interfaces for non-expert scientists to browse through and retrieve data. Solutions to these challenges should be applicable to systems that support either type of activity.

[1]The term 'local' is only used as an indicator of scale, with no connotations about the proximity of the scientists involved in the study or the location of the data used in the study.

The general theme of this paper is managing local experimental studies. We introduce the term 'desktop *Experiment Management System*' (EMS), to describe a system that supports such activities. Such a system, which includes a Database Management System (DBMS) as one of its components, will be the only tool that a scientist uses to manage his/her experimental studies. It will support the scientist in the design of the study, communicate with the appropriate environments from which the data for the study is collected, and store and manage that data. The operational environment of experimental studies has the following unique characteristics that place certain demands on what the desired functionality of an EMS is:

(i) Each experimental study goes through several stages that are quite different from each other. To avoid overburdening the scientists, who should not have to be experts in database management, the EMS should provide a uniform interface that can be used in the diverse activities related to all these stages.

(ii) In today's scientific laboratories, where experimental studies are conducted without much computerized technical support, communication among collaborating scientists is quite interactive. To facilitate the same mode of communication when computer technology is used, the EMS should provide an efficient and natural user interface that resembles, to the extent possible, the way scientists interact among themselves.

(iii) Many experimental studies are in need of generating data in multiple diverse ways and using existing data from multiple sources. The EMS should be capable of communicating with all these heterogeneous information sources and integrating the data that they provide without requiring much detailed knowledge from the scientists.

Providing the above functionality presents many problems to today's technology. These problems are further exasperated by the complexity of the structure of the data and experiments manipulated by the EMS.

In this paper, we identify some of the fundamental issues that must be addressed in designing an EMS so that its goals may be achieved. An important aspect of this work is a proposed EMS architecture that is centered around the extensive use of conceptual schemas, which express the structure of information in experimental studies. Schemas are called to play new roles that are not usually found in traditional database systems. We provide a detailed exposition of these new roles and elaborate on the implications of such schema use. Specifically, we describe certain characteristics that the data model of the EMS must have in order for schemas expressed in it to successfully play these roles. An interesting side result of the above effort is the development of an abstraction of the set of activities performed by experimental scientists throughout the course of a study, on which the details of the proposed EMS architecture are based. Following the above general principles on how to support the management of experiments, we have undertaken an effort to develop a desktop EMS that achieves the desired goals. We present the specifics of our approach in the later part of this paper. In particular, we describe the salient features of the data model that we have developed for the EMS justifying their inclusion in the model by the needs of experiment management. We also discuss a case study where schemas expressed in that model played some of the new roles mentioned above in the context of some scientific experiments.

As a reference point that can be later used to illustrate the various issues raised in the paper, we describe a very simple experimental study. Simulation is being used to model the effect of weather on plant communities. Its input consists of weather parameters, which are humidity and wind speed and direction, and characteristics of a plant community, which are the locations of all plants and the number of leaves, height, and type (e.g., corn, wheat) of each plant. Its output is the vegetation temperature, one temperature value for each plant. The simulation itself takes into account the relative placement of the plants and all the physical laws on how each type of plant reacts to the weather conditions based on its environment. An EMS used for this study will allow scientists to design the input and output structure of the experiments, invoke executions of the simulator, store the collected data, and submit queries on the experiment results.

Among all types of schemas, we only deal with conceptual/logical schemas in this paper. Hence, we often use the plain term 'schema' instead of the full term 'conceptual schema'. Also, we imagine that most users of an EMS will be scientists, researchers, or technicians working in a laboratory. For the purposes of this paper, we make no distinctions among the above types of experimentalists, so we use all the above terms (including the term 'user') indistinguishably to refer to the generic user of an EMS. Finally, databases containing the data associated with experimental studies are called 'experiment databases'.

This paper is organized as follows. Section 2 describes a common life-cycle that underlies most experimental studies. Section 3 outlines the functionality that an EMS should provide to its users and proposes an architecture that we have adopted for such a system that we are currently developing. Section 4 identifies some new roles that conceptual schemas are called to play in the context of an EMS. The characteristics that the data model should possess in order for its schemas to play these roles are also identified in this section. Section 5 discusses the salient features of the Moose data model that we have developed for experiment management. Section 6 contains a brief description of a case study where some of the tools that we have developed for manipulating Moose schemas were used in an experimental study. Finally, Section 7 summarizes our approach for experiment management and discusses the future directions of our work.

2. Life-Cycle of Experimental Studies

To achieve its goals, an EMS will use conceptual schemas for various activities that are important throughout the course of an experimental study. From discussions with scientists from different disciplines, we have concluded that these activities are common to most experimental studies. We use the term *life-cycle of an experimental study* (or simply *experiment life-cycle*) to denote the entire set of these activities together with the way scientists iterate over them during such a study. In this section, we describe the different stages of that cycle, so that the details of the different roles of schemas throughout the cycle can be explained later. We should emphasize at this point that the experiment life-cycle that we describe only captures the activities involved in conducting the experiments and not those involved in setting up the appropriate experimentation environments. For example, in the case of simulation studies, it does not capture the programming task of developing the simulator, but it does capture the task of executing the simulator with a specific set of input parameters.

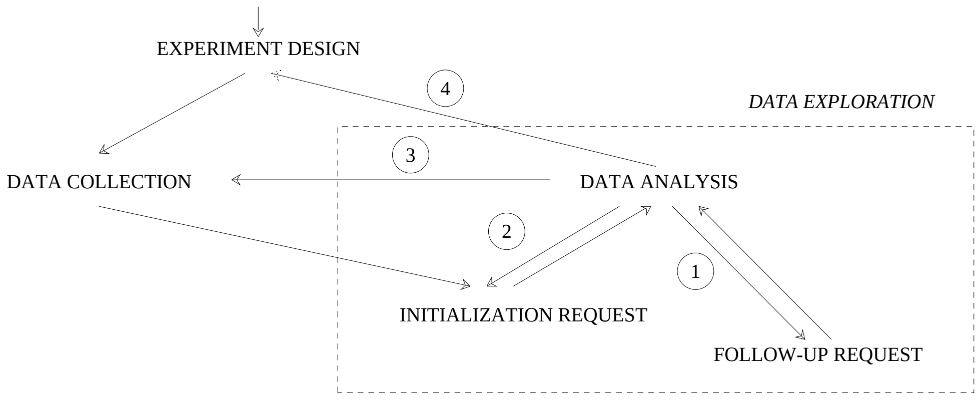

Figure 1: Life cycle of an experimental study.

A pictorial abstraction of the experiment life-cycle is shown in Figure 1. It essentially consists of multiple loops traversed by the researcher multiple times in the course of a study. In the figure, the following stages can be identified:

Experiment Design: In this stage, the *experimental frame* of a study is laid out [Zei76], that is, the structure of each experiment is defined. The experimental frame determines the variables that will be controlled in the experiments and defines what will be measured as output. For the example of the plants experiment of Section 1, this stage consists of identifying the input and output parameters of the simulator and their relationships based on their semantics. Properly designing the experiments is the most crucial aspect of an experimental study. A satisfactory design is rarely achieved in a single attempt. This process undergoes many iterations, usually interleaved with the execution of some experiments and analysis of the obtained data, before the design reaches its final form.

Data Collection: In this stage, experiments are actually conducted. The researcher specifies the experiment set-up and the precise values of all the input parameters to the experiment, and the relevant output data is then collected. The data can either be distributed to some or all of the scientists involved in the study or it can simply be stored for later use. Simulating a specific plant community given certain values for its characteristics and the weather conditions is an example of an action in this stage for the plants experiment.

Data Exploration: In this stage, the researcher studies the collected data to draw conclusions about the subject of the experiment. As shown in Figure 1, there are three types of actions that the scientist may perform on the data, which are described separately.

- *Initialization requests:* Whenever scientists start to explore a new vain of thought in an experimental study, their first request on the collected data is very similar to a conventional query in traditional database systems. It references all properties of the phenomenon or system under study that are expected to remain unchanged throughout the exploration of the new idea. In principle, such a request needs to deal with the full experimental frame of the study and must include specifications of the values of many parameters, the relationships between several others, and some indication of what should be retrieved. A conceivable initialization request for the plants experiment may be for the final temperatures in corn communities with given plant characteristics and weather conditions when the distance between any two plants is less than 1 meter. In most cases, due to the amount of information that must be specified, posing such queries is a time consuming process.

- *Data Analysis:* After receiving the requested data, scientists analyze it based on domain-specific knowledge that is relevant to the studied phenomenon. Occasionally, the analysis is not based on the data retrieved by the scientists' requests, but on the output of some further processing on it. Such processing is invoked by applying domain specific operators to the data, e.g., measuring the intensity of an image, extracting the statistical properties of a time series, or obtaining the difference between two functions.

- *Follow-up Requests:* Based on the results of the analysis of some obtained data, quite often scientists pose new requests that are very similar to the previous ones, having the answers of the latter as a reference point. This is due to the predominantly exploratory nature of experimental science, which forces scientists to navigate through a multi-dimensional space of parameters that captures the behavior of the observed phenomenon. As an example of a follow-up request, after the above initialization request in the plants experiment, scientists may ask for the same but for distances less than 2 meters. The difference between the two requests is only the value in the selection clause on distances. Follow-up requests represent the most common form of interaction in the course of a study. It is therefore extremely important that such requests be efficiently and conveniently expressed.

If Figure 1 is seen as a directed graph, then it is clear that all graph nodes except for data analysis have outdegree 1, i.e., the successors of the corresponding stages are predetermined. On the other hand, after data analysis the study may move to any of the other stages. Each one of the corresponding arcs closes one of the loops of the life-cycle mentioned earlier. These loops can be totally ordered based on their frequency in the life-cycle, i.e., based on how often the scientist follows the corresponding arc after data analysis. That ordering is indicated in Figure 1 by the numbers labeling those arcs, where 1 indicates most frequent and 4 indicates less frequent. For example, it is more likely that a scientist will pose a follow-up request after analyzing some data than that he/she will redesign the experiment.

It is worth noting at this point that the separating line between the data collection and data exploration stages is rather hazy, in the sense that data exploration may involve hidden and not explicitly requested data collection. When a scientist is studying a phenomenon, whether a specific piece of information has already been collected or needs to be collected via an experiment is irrelevant. Thus, some requests in the data exploration stage may generate orders for data collection. For example, consider the initialization request on the plants experiment mentioned above. If corn communities have never been simulated with the given plant characteristics, then simulation may be automatically initiated as a result of this request to obtain some relevant data.

We should also mention that the above life-cycle represents experimental studies that are conducted either by individual scientists or by teams of scientists. In the latter case, most stages of the life-cycle involve communication among the collaborating scientists. This communication can be in the form of actual real-time interactions for decision making (mostly in experiment design and data analysis) or in the form of concurrent actions of multiple scientists, the results of which are later integrated together (mostly in data collection and initialization and follow-up requests).

Having presented the general structure of the life-cycle of an experimental study, we would like to use that cycle to clarify the distinction between the two types of activities mentioned in Section 1, i.e., managing and distributing the primary data for a global project and managing a local experimental study. The primary focus of the former activity is in the first two stages of the life-cycle, i.e., experiment design and data collection, with minimal or no interleaving between them. For example, after the initial design stage of measurements of the Eos project, satellites will be launched to start collecting the prescribed data without much interference. Requests for the collected data by interested scientists will be mostly for small subsets of data that are related to a small geographic region, period of time, or specific phenomenon. Although it is possible for a scientist to submit multiple such requests in a given session, especially in a browsing mode, the complex iterations that the full data exploration stage implies will not be required in general. After the data is identified and distributed, the scientist(s) involved will perform their study without any further interaction with the central repository. This study of the obtained data will be an activity of the second type mentioned above, involving the full life-cycle of Figure 1. Note that in a global project, experiment design and data collection are performed by a carefully chosen team of domain scientists and database administrators, whereas data exploration (in the form of simple browsing and retrieval) is performed by any scientist in the field. On the contrary, in a local study, the full life-cycle is performed by the same scientist(s). Clearly, the above distinctions between the two types of activities are not absolute. We believe, however, that in general there are some characteristic differences between them, which we hope have been exposed by the above comparison in the overall framework of the experiment life-cycle.

In the rest of the paper, we focus entirely on the second type of activity, i.e., on managing experimental studies conducted by individual researchers or small teams of them. Based on the above discussion and the structure of the experiment life-cycle, we examine some of the technical challenges faced by attempts to develop EMSs to support such activities. We then propose some solutions that we have adopted in our own efforts, which are centered around the versatility of conceptual schemas and their usefulness in a wide variety of tasks.

3. Functionality and Architecture of Experiment Management Systems

Current database technology provides very primitive tools in the hands of scientists involved in experimental studies. All stages in the experiment life-cycle are viewed as distinct from each other with no or minimal communication among them. The transfer of data and the transition from one stage to the next are to a large extent 'manual'. Thus, many scientists end up using flat files to store the data of their experiments. For example, the following senario is quite common. Collected data is stored in files with cryptic names like 'out.1.100.13.7', which usually encode the values of the input parameters to the experiment. At data analysis time, application programs in conventional languages are written for every type of desired output. These programs have to look into the mass of files containing the relevant data, extract the useful information, and format it so that it is presented to the scientist in a meaningful way. Moreover, searching for the relevant data each time cannot be performed associatively (by the desired values of some parameters) but only by name, i.e., the given names of the files. Clearly, this process is very unnatural, tedious, error-prone, and requires constant exposure of the scientist to the specific set of data, because the meaning of the various parameters is easily forgotten.

The role of an Experiment Management System (EMS) should be that of an agent between a scientist and a phenomenon under study. An EMS should provide the desired functionality for managing and analyzing data produced in experimental studies and overcome the inadequacies of today's technology. Such a system should be a single, integrated, tool that scientists can use throughout the life-cycle of an experimental study to effectively control and manage all aspects of the experimental process and the generated data, i.e., it should satisfy requirements (i)-(iii) of Section 1.

In order to achieve the above functionality, an EMS must be capable of both managing stored data and communicating with one or more *experimentation environments* (where experiments can be run) and other EMSs and DBMSs to obtain new data. Much like in a heterogeneous database system, given a scientist's request, the EMS will first identify the experimentation environments and/or systems that are related to the request. It will then divide the request into pieces, translate each piece into the language of its target environment or system, and submit it for processing. In the end, the EMS will collect all the responses, generate one integrated result out of them, translate that into the appropriate user-level representation, and return it to the scientist who posed the request.

An EMS should be able to communicate with other EMSs and DBMSs that manage data of interest already collected as part of other studies, so that duplication of effort is avoided. It should also be networked with several experimentation environments due to the very nature of experimental studies. In many cases, in order to investigate a phenomenon, or develop a new system, experiments under various control levels are performed. At one end of the spectrum are fully controlled experiments in which the system is simulated on a computer. In contrast to this, in the laboratory, where the environment can be controlled but the system has a life of its own, the observer has only partial control over the experiment. Finally, no control can be exercised when the real world is observed. An EMS that provides a cohesive interface to a range of experimental environments, which have been independently developed, possibly to solve problems of diverse scientific fields, has many advantages: a) transitions are smooth from one environment to the other, b) experimental data from different sources are analyzed in a single framework, and c) the EMS serves as a bridge between different experimental disciplines. An EMS with all the above capabilities will provide the richest possible support to scientists who will be able to flexibly use unlimited amounts of information to further their own research.

Another important feature that an EMS must have to achieve the desired functionality is that it must be capable of blurring the distinction between data collection and data exploration (data requests in particular) if the scientist so desires. This would conform to the natural vagueness of the separation between these two stages in the experiment life-cycle (Section 2). In particular, the scientist should be given the freedom to request data without any knowledge of whether it has already been measured and recorded or not. Depending on the situation, the EMS should decide whether to simply retrieve the data from its database, or initiate some action outside of the system.

Driven by the need for a tool to provide the kind of support described above in our own experimental studies, we have initiated an effort to develop an EMS at the University of Wisconsin. Figure 2 presents the architecture of that system, which we believe reflects the needs of a wide range of experimental studies. In addition to a component for the traditional query and storage services provided by a database system (*Core DBMS*), the EMS under development has an active component, which coordinates the interaction between the user and the experimentation environments (*Experimentation Manager*), and an analysis component for the stored data (*Output Analyzer*). The user interacts with the database system via intuitive language and graphical interfaces (*User Interfaces*). Finally, a variety of experimentation environments are coupled to the EMS via a component that translates data from its representation in the experimentation environments to its representation in the EMS and vice versa (*Data Translator*).

We believe that a system developed based on the above architecture will have achieved its most important goals. It will provide an integrated environment to scientists that, unlike current practice, will feature a uniform interface that may be used for managing the entire life-cycle of an experimental study. Moreover, it will allow the design and execution of experiments and the access to scientific data to be done in ways that resemble as much as possible the way scientists interact among themselves using pencil and paper.

A fundamental premise of our effort has been that the above cannot be achieved unless the EMS provides a uniform and natural interface for all stages of the experiment life-cycle (item (ii) in Section 1). The most important piece of information that is necessary in all these stages is the conceptual schema of the database related to an experimental study. Thus, it is only natural that our approach is 'schema-centric', where schemas are used in many roles (some of which are quite unique) throughout the experiment life-cycle, providing a common foundation for all types of interactions between the scientists and the EMS. This importance of schemas has also been the reason why our first priority has been to obtain a better understanding of schemas and their use, and to develop the schema support of the EMS, on top

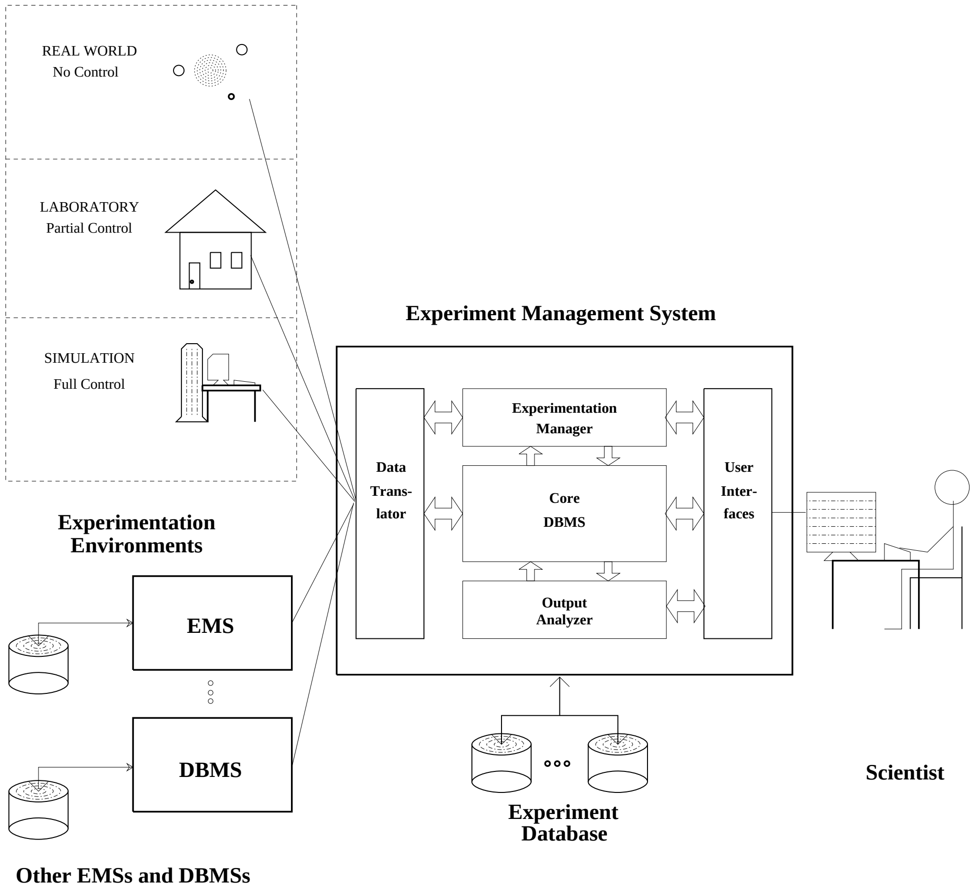

Figure 2: Architecture of an Experiment Management System.

of which the rest of the system will be built. The results of this first phase of our work are described in the following sections.

In addition to our own effort to develop an EMS, some research laboratories have also been engaged in similar work trying to provide database support for scientific data. Examples include the 'Laboratory Notebook' project in the Los Alamos National Laboratory [Nel90] and the effort to develop data management tools for scientific applications in the Lawrence Berkeley Laboratory [MF91, SM91, MS92]. Several aspects of our effort are found in at least some of these projects: schemas play several important roles, and intuitive (usually graphical) user interfaces are developed so that scientists may use them without much database expertise. On there other hand, several differences exist as well. The most important of them is that, to the best of our knowledge, our effort is the only one that attempts to provide a single tool for all stages of the experiment life-cycle. The other projects focus primarily on experiment design and initialization requests, which are similar to activities in traditional database management. Supporting data collection or the complex iterations of data exploration is not part of the functionality of the systems developed in these projects. Less significant are differences in the choice of data model (they are based on the relational or the extended entity-relationship model, whereas we have developed our own object-oriented data model), and in several other system aspects, on which we do not elaborate.

4. Roles of Conceptual Schemas

In this section, we describe the roles that the schema plays in an EMS for each stage of the experiment life-cycle. In addition, we outline the features that a data model should have in order for schemas expressed in that model to play those roles.

4.1. Conceptual Schemas in the Experiment Life-Cycle

By definition, schemas capture the structure and constraints of the data that is recorded in a database so that only valid data is accepted for storage. In most current database systems, schemas are used primarily for the above purpose. They are defined and altered by the database administrators, but cannot be manipulated or updated by end-users. Such users may only consult the database system for information describing details of schemas, which is provided by help facilities. Thus, it is the user who initiates the flow of schema-related information out of the system. The DBMS itself is *passive* and only responds to user requests. The above state of affairs is considered adequate given the current roles played by schemas. In fact, the schemas of many databases are relatively small, so with frequent use, even memorization of the relevant parts of the schema by the user is common.

Although in traditional settings the above use of conceptual schemas is considered satisfactory, in an EMS it is not. For an EMS, the schema is a useful tool for many more activities than in traditional DBMSs. In addition, by the nature of scientific studies, most user interactions with the system are in the form of ad-hoc queries, whereas in traditional settings, running prepackaged application programs is much more common. In combination with the complexity and size of typical schemas of experimental studies, this makes the fact that the EMS has accurate knowledge of the schema much more valuable than it is in a conventional DBMS. Thus, the schema is called to play new roles in the context of an EMS. Accordingly, the EMS is forced to provide enhanced functionality compared to a traditional DBMS with respect to manipulating the schema and become *active* by taking the initiative in presenting the schema to the scientist.

Whereas in a conventional database the schema captures the structure of the data in the database, in experiment databases, the schema also captures the structure of the experiment itself. This is a side-effect of the effort to describe the structure of the data: in order to organize the data in a meaningful way, the design of the experiment is essentially represented as well. For example, the schema of the plants experiment contains the various input and output parameters and their relationships to plant communities, which is precisely the information required to capture the design of the entire experiment. Based on this interpretation of its contents, the schema is called to play two new major roles in an EMS, in addition to its traditional roles:

(R1) In its first new role, the schema becomes the formal document describing the experiment. This is important for both individual and collaborative studies. Designing experiments, modifying earlier designs, describing experiments to others, integrating pieces of experiments into larger studies, and other activities that are usually based on arbitrary, and quite often free-form, descriptions of experiments are now based on the conceptual schemas of the corresponding databases. In fact, in the first stage of the experiment life-cycle (Figure 1), the old notion of database design can now be seen in the new light of generalized experiment design.

(R2) In its second new role, the schema serves as the template for specifying data and experiments. Such specifications are useful both in interactions between scientists and in interactions between a scientist and the EMS. (Specifying data is not a new idea; although it has not been extensively used in commercial systems, it has been proposed and studied in the context of many research prototypes, e.g., [AGS90, BH86, Fog84, GGKZ85, KM89, P$^+$92, RC87, WK82, Zlo77].) The ability of the schema to play this role is important in the data collection and data exploration stages (Figure 1). The system itself prompts the user with the schema, who then manipulates it appropriately for specifying query restrictions or for displaying query answers.

From the above description of the two roles, it becomes clear that the use of the schema spans all stages of the experiment life-cycle, which is fundamental to providing an integrated tool with a uniform interface to scientists. It also becomes clear that the conceptual schema undertakes these two roles not only within the EMS, but also in interactions between collaborating scientists as well. This can prove extremely important in the future, where multidisciplinary studies with large numbers of scientists participating will become more common [HL92].

4.2. Necessary Data Model Characteristics

In order for schemas to play the above two roles successfully, they have to be expressed in a data model that has the following three characteristics. First, for R1, the data model needs to be of high expressive power. Scientific experiments have quite complex structures, so the data relationships that must be captured in experiment databases are quite complex as well. The relational model is in general inadequate due to its simplicity. Its unique semantic primitive, the relation, is not powerful enough to express every aspect of an experiment design. The much richer object-oriented and semantic data models [CM90, ZM89] are the only serious candidates for such databases. Among other features, such data models offer primitives that can be used to represent complex objects (parts-subparts), collection objects (sets), and class hierarchies with inheritance, which are very common in experiments.

Second, for both R1 and R2, the semantic primitives of the data model must be closely related to notions that scientists are currently using in their approach to experimental studies. A data model that is developed based on database expertise while ignoring the status quo in today's scientific laboratories is doomed to fail. Scientists must feel comfortable with the primitives of the data model so that they do not have to establish complicated mental mappings from their current way of thinking to that enforced by the model. As mentioned above, it is desirable for the data model to have high expressive power, but this should not be achieved at the expense of natural expression. The primitives of the data model should reflect the experience of scientists so that the complex data relationships found in experiment designs can be captured in a natural way.

Third, for both R1 and R2, schemas in the data model must have a succinct representation so that they are easily understood by scientists. Traditional text-based data definition languages may not be the most appropriate tools for scientists to use for schema specification. SQL has quite a long and slow learning curve, and even for experienced users, writing very complex queries is not straightforward. It is doubtful that learning how to use a similar, but more complex, text-based language for a very expressive data model is the best use of scientists' time. Intuitive graphical representations of schemas, supported by user-friendly interfaces, will be of much more use to the scientific community.

From the above arguments, one may conclude that an EMS should have a graphical user interface that can deal with large and complex object-oriented/semantic schemas in a natural way, allowing the user to manipulate the schemas for multiple purposes. Clearly, the demand for the first characteristic in a data model is not unique to scientific experiments. Many other applications have similar needs, which have driven the numerous research and commercial efforts to develop systems based on semantic and object-oriented data models. The demand for the second and third characteristics, however, is not so common. In most DBMSs, schemas are manipulated only by database administrators and complex queries are packaged in easy-to-invoke applications written by professional programmers, who are very experienced specialists in their respective fields. In an EMS, on the other hand, we want the scientists themselves to be able to interact with the system as both database administrators and sophisticated end-users. Otherwise, much of the promised power of EMSs will be jeopardized. Thus, the need for data model primitives that are natural to scientists and for intuitive representation of schemas manipulated by easy-to-use tools is much more pressing in EMSs than, perhaps, other types of DBMSs and has received more focussed attention in our work.

In the following sections, we describe the data model that we have developed as part of our EMS effort together with some key features of the graphical interface that supports schemas in the model.

5. Moose: A Data Model for Scientific Experiments

The EMS that we are developing is based on the *Moose* (*M*odeling *O*bjects *Of* *S*cientific *E*xperiments) object-oriented data model [IL89]. Although Moose is targeted for experimental data management, it is applicable in much more general settings as well. The salient features of Moose are described below.

5.1. Semantic Primitives of Moose

We first present the semantic primitives of Moose that define its expressive power. In the description, we put more emphasis on features that are not common among the already existing semantic and object-oriented data models, justifying their inclusion in Moose by the needs of experiment management. Thus, we illustrate why we believe Moose satisfies both the first and the second desirable data model characteristic mentioned in Section 4.2.

Moose supports the notion of an object, which is quite intuitive to scientists because most often objects are used to represent physical entities that are relevant to experiments, e.g., the planet Jupiter or the part of the E. coli genome known as K-12 strain MG1655. Every object is assigned a unique object identifier and belongs to possibly multiple *classes*, inheriting properties from all of them. A class represents a set of objects having the same structure and the same properties. There are four system supported object classes, called *base classes*: integers, floats, character strings, and booleans.

The *extent* of a non-base class, i.e., the objects that are known members of the class, is explicitly stored in the database. This allows objects that are currently not part of any experiment design, i.e., that are not associated with other, higher level, objects, to still be stored and manipulated by the user. For example, in a simulation study of plant growth, one may want to introduce into the system a new variety of corn. Although, there are no experiments that have been run with this type of corn, nevertheless it is important that information about it is stored in the system, so that it is available later when the scientist decides to run experiments with it. In addition, 'inventory queries' of the form 'What types of corn do I have at my disposal?' are possible. The above needs are so common in experimental studies that having the scientist explicitly request the maintenance of the class extent for each class would require a significant effort, since it would have to be done for most classes in the schema.

In many experimental studies, object collections are often reused several times during the course of the study, e.g., a specific plant community. In addition, they are often associated with other pieces of information, which may or may not depend on the objects in the collection, e.g., the number of objects in the collection or some name given to the collection, respectively. To serve the above needs, collections of objects are individual objects themselves in Moose, carrying all the characteristics mentioned above. This uniform treatment of atomic and collection objects results in an economy of scale and makes sharing of collections and expressing properties of collections very natural. Otherwise, additional object classes would have to be defined, cluttering the schema and moving it further away from the usual intuition of scientists. There are four kinds of collection objects supported in Moose: sets, multisets (bags), indexed-sets (a generalized form of arrays), and sequenced-sets (a generalized form of lists).

Each class in a Moose schema may be associated with many other classes, capturing a variety of relationships that may exist between the objects of the corresponding classes. Similarly to most semantic and object-oriented data models, Moose supports two major types of relationships: *is-a* relationships and *part-of* relationships. The former capture semantic relationships whereas the latter capture structural relationships between objects of the participating classes. Specifically, is-a relationships relate classes to their subclasses (specializations) and vice versa, whereas part-of relationships relate objects to their parts and vice versa. Every part-of relationship is associated with a label, which serves the same purpose as an attribute name in relational DBMSs. For this reason, we occasionally use the term 'attribute' to indicate part-of relationships. The direction of a part-of relationship is from a class of objects to the class of their parts. Every part-of relationship, however, essentially captures a function and its inverse and can be explored in both directions. Therefore, it is associated with two labels. Quite often, one or both of these labels is equal to the name of the range class of the relationship traversed in the direction corresponding to the label, e.g., the utilization of a 'cpu' is a 'utilization'. Whenever this is the case, we omit the label from the relationship declaration.

Two more types of relationships are supported by Moose to capture specialized associations of collection objects. A *set-to-elements* relationship connects a collection class to the class of elements in the collection, e.g., from the class of plant communities (sets of plants) to the class of plants. Exactly one such relationship must exist for each collection class. The need for this relationship is an immediate consequence of the need to support collections as first-class objects. An *indexing* relationship connects an indexed-set class to the collection class indexing it, which is the *key-set* class of the relationship. Its semantics is that, each member of a key-set is associated with exactly one member of the indexed-set. Exactly one such relationship must exist for each indexed-set, except for those that are indexed by the natural numbers (i.e., those that are arrays in the traditional sense), for which such a relationship is implicit. Scientists need this relationship to express functional dependencies from the members of the key-set class to those of the indexed-set class. Such dependencies arise very often when the same parameter of the input or output of an experiment takes on a different value for each member of some collection related to the studied phenomenon or system, e.g., every distinct plant in a community has a different temperature. The parameter values (e.g., the temperatures) form the indexed-set and the collection (i.e., the plants) forms the key-set of this relationship. Through that, the parameter value associated with an object from the indexed-set is directly available. In the absence of this type of relationship, for the same semantics to be captured, either auxiliary object classes would need to be defined, or the scientist would have to assign an integer number to each element in the key-set so that regular arrays could be used, which are supported by most systems. In the first case, again the size of the schema would increase with classes that play no substantial role in the scientist's experiment design. In the latter case, the scientist would have to constantly use some unintuitive numbering to be able to indirectly associate elements to parameter values.

Finally, in Moose, part-of relationships (and less often other relationships as well) can be declared as *context-dependent* [WTM$^+$92]. A relationship of this type may be used to capture an association between a pair of object classes that depends on a third class as well. For example, in a study evaluating the performance of networks, a network site may be associated with a different job arrival rate in each experiment. This may be captured by a context-dependent relationship between the class of sites and the class of arrival rates, with the class of experiments serving as the context. By definition, many relationships between objects in experimental studies are context-dependent on experiments. By having the ability to directly represent such relationships, scientists are able to design their experiments more naturally than otherwise.

Moose allows objects that are parts of a given object or instances of a given class to be defined either intentionally or extensionally. Specifically, the parts of an object do not have to be explicitly specified by the user. Moose supports the notion of a *virtual attribute* whose contents can be derived by the system through some computation associated with the attribute. Such computations are expressed in the form of rules that are based on the query language of Moose (whenever possible) or in some general computationally complete language among those supported by the system (whenever necessary). Virtual attributes are especially useful to scientists for specifying aggregate computations over the members of collection objects. For example, every set of plants may be associated with a virtual attribute whose value is always calculated by counting the number of plants in the set. Given that the task of almost all experimentalists is the statistical study of some phenomenon or system, the implicit computation of aggregates as virtual attributes is an important tool. Moreover, the power of this feature goes beyond aggregate values and can be used for other purposes as well. For example, the entire output of an experiment can be considered as a virtual attribute that depends on the experiment input and the contents of which are computed by conducting an experiment.

Similarly, the membership of a class does not have to be explicitly specified by the user. Moose supports the notion of a *virtual class* whose membership can be derived by the system through rules associated with the is-a relationship between the class and some superclass of it. The importance of virtual classes can be realized by examining the experiment life-cycle (Figure 1). As a scientist explores the results of the conducted experiments, important characteristics of objects used in the experiments are identified. For example, a special behavior may be observed when the arrival rates in all sites of a network are the same. Upon such a discovery, it is common for scientists to give a special tag to such networks, e.g., call them 'homogeneous networks', and put some further emphasis on investigating their behavior. In the context of an EMS, the equivalent steps are for scientists to define the virtual class of homogeneous networks as a subclass of networks and to associate the appropriate rule defining the members of the subclass with the corresponding is-a relationship. A nice side-effect of this action is that any networks that happened to be homogeneous and were used before the scientist realized the importance of that subclass implicitly become its members, without any additional work.

Both types of implicit definitions remove significant work from scientists and enhance their ability to express complex relationships among classes. In addition, for all definitions expressed in the rule language of the system, inferences are made without explicit instructions from the users.

Finally, Moose supports many types of user-defined structural constraints that may be used to control sharing among objects. A relationship may be one to one (referred to as single-valued, non-shared), one to many (multivalued, non-shared), many to one (single-valued, shared), or many to many (shared, multivalued). Moose also has a constraint language, which may be used to express more complex structural constraints than the above. Such constraints express important aspects of the semantics captured by schemas. They are necessary in both general DBMSs and EMSs, which may use them to ensure the integrity of the stored data.

5.2. Graphical Representation of Moose Schemas

As mentioned in Section 4.2, the third important characteristic that a data model should possess in order to be useful in an EMS is that its schemas should have a succinct and intuitive representation, so that scientists who are non-experts in database management can manipulate them without much effort. This has been one of the major concerns throughout the development of Moose.

The result of our work in this direction is that Moose schemas can be defined graphically and manipulated by appropriate actions directly on the iconic representations of their primitives. Specifically, every Moose schema has a straightforward directed graph representation. Every node in the graph represents a class of objects and is labeled by the class name. Base classes are represented as ellipses, to be easily distinguishable from the rest, while all other classes are represented as rectangles. In addition to the corresponding class name, nodes representing collection classes are also annotated with a special symbol identifying the type of the collection, e.g., for sets, for multisets, [] for indexed-sets, and () for sequenced-sets.

Arcs in the graph capture the various types of relationships supported by Moose. Part-of relationships are denoted by solid arcs, is-a relationships are denoted by dotted arcs, set-to-elements relationships are denoted by double solid arcs, and indexing relationships are denoted by zig-zag arcs. Part-of arcs are labeled with the name of the associated relationship, unless the label is the same as the name of the class at the head of the arc, in which case it is omitted from the graph. Context-dependent arcs are annotated with the name of the context class as well. Finally, the structural constraints mentioned above that control sharing among objects can also be represented graphically. All four combinations of such constraints are shown in Figure 3 for part-of arcs; the same constraints are captured similarly for the other appropriate arcs.

As an example of the graphical representation of Moose schemas, Figure 4 shows some possible schema for the plants experiment of Section 1. In addition to the original features, we also include the notion of homogeneous plant communities, which

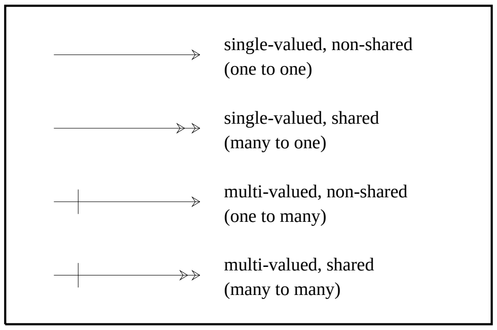

Figure 3: Graphical representation of structural constraints.

form a subclass of plant communities. The rule r associated with the corresponding is-a arc captures the precise definition of homogeneous plant communities, e.g., those where all plants are of the same type. Also, administrative information like the date of the experiment and the amount of time consumed by the simulator (its cost) are shown as attributes of the experiment. (We should emphasize that alternative schemas do exist for the plants experiment, some of which would possibly be more flexible than the one presented but also more complex. The above was chosen as a good trade-off between simplicity and flexibility.) In Section 6, another complete Moose schema is shown graphically, in the form of a screen-dump of a prototype that we have developed for part of the user interface of the system.

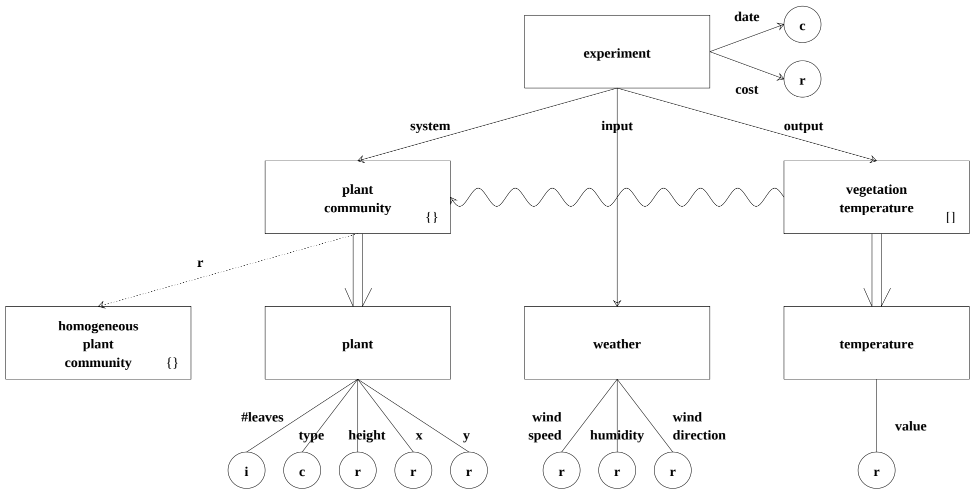

Figure 4: Graphical representation of schema for the plants experiment.

5.3. Moose Query Language

The query language of Moose is very similar to SQL and also has the flavor of other declarative object-oriented languages [ASL89, BCD89, CDV88, KKS92, KL89]. Since the query language is not the focus of this work, we are not describing it in any detail. We only want to point out that the fundamental construct in the language for traversing objects is the *path expression*. Path expressions connect objects in some class to other objects that are directly or indirectly related to them. A novel and powerful aspect of path expressions in the Moose language compared to other such languages is that they can include relationships of all types, and not only part-of ones. Path expressions are closely associated with the graph representation of a Moose schema, since they essentially indicate paths in that graph. Based on this characteristic, a graphical query language that is closely related to the textual one is also under development. Given the focus of our effort, we expect that the graphical language will be the primary means of interaction of scientists with the system.

5.4. Summary

Based on the above brief description of the features of Moose, we believe that it satisfies the three necessary characteristics described in Section 4.2. First, it supports a rich set of data types and semantic relationships between data that can be used in arbitrary combinations to capture the complex structures of experimental data. The various kinds of constraints can be used to ensure the integrity of the data, while the ability to define virtual attributes and classes (quite often through the use of rules) removes much burden from the user. Second, the semantic primitives of Moose have been chosen based on the needs of scientists. Especially the primitives that are not usually seen in other object-oriented or semantic data models capture specific notions that are very intuitive to most scientists. The ability to use these same notions in the scientists' interactions with an EMS should make the system a much better and friendlier tool, better serving the needs of its users. Third, the graph corresponding to a Moose schema is a much more compact and intuitive representation of the schema than any other form, offering to scientists a better means to express their experiment designs. In addition, the development of a graphical user interface becomes possible, enhancing the usability of the system even further.

6. Using Moose for Experiment Management

Due to our firm belief in early prototyping, our study of experiment management has proceeded in parallel with the development of a prototype EMS, where the findings of our work are implemented so that they can be tested and validated. The goal of our effort is for the EMS to provide the appropriate technical support that will take advantage of the rich set of semantic primitives of Moose and its nice graph representation to become a versatile tool for scientists. In this section, we focus on a piece of the user interface of the system that has already been built, and describe its use in the context of an experimental study. In particular, we describe the graph editor of the system, which can be used to manipulate arbitrary types of graphs, and most importantly Moose schema graphs. The graph editor is a key part of the whole user interface of the EMS. This is due to the graph representation of Moose schemas, which transforms any schema manipulation to graph manipulation independent of the role played by the schema (Section 4).

The main difficulties in graph editing arise from the fact that Moose schemas for scientific experiments tend to be very large and can form an inscrutable maze of boxes and lines on the screen. Therefore, the key features that have been included in the graph editor deal with making 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 [ILH92]. The above features are expected to prove very useful in supporting all aspects of schema use mentioned in Section 4. This expectation is justified by the results of our exposing the graph editor developed to 'real' users, i.e., domain scientists, for experiment design (first stage in Figure 1). In all such collaborations, scientists from other disciplines have used the schema in its first new role, i.e., as a formal document describing experiments (Section 4) and the feedback obtained has been very encouraging.

Our work with John Norman from the Soil Sciences Department at the University of Wisconsin is one example of such collaborations. The main emphasis of his research is on simulating the growth of plants based on various environmental, soil, and ecological parameters. The primary tool in his studies is the *Cupid* model [ILH92], a Fortran program that simulates the necessary plant growth processes. The Cupid group has been using the graph 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.

Cupid has been an excellent testbed for the capabilities of Moose and the graph editor because it has a complex structure and generates very large schema graphs. It simulates numerous processes that are parameterized, and therefore a large number of parameters need to be specified to characterize its input and output. Typically, about a hundred parameters are input to the model for any specific application, whereas the output variables number in the several hundreds. Our collaboration with the Cupid group has shown that the graph editor we have developed can serve as a tool to organize the large amounts of data that Cupid manipulates. The schema for the input part of the Cupid model has been completed and contains more than a hundred object classes. It is shown in Figure 5 as a screen dump of the graph editor developed.

The feedback received from the Cupid group, which used Moose and the graph editor to design the schema of Figure 5 has been very encouraging [ILH92]. The main benefits expressed follow quite closely the analysis of desirable features of the data model and the system described in earlier sections. Casting the Cupid data in an object-oriented structure captured the complex array of data combinations that were part of the model in a natural way and made further modifications and enhancements of it easier. The graphical representation of the Moose schema was instrumental in allowing the members of the Cupid group to understand Moose relatively quickly and then use it for the experiment design. Finally, the resulting Moose schema played its R1 role well: it served as a clear documentation of the input structure of the Cupid model and proved very useful in communicating its details to other scientists outside of the main group.

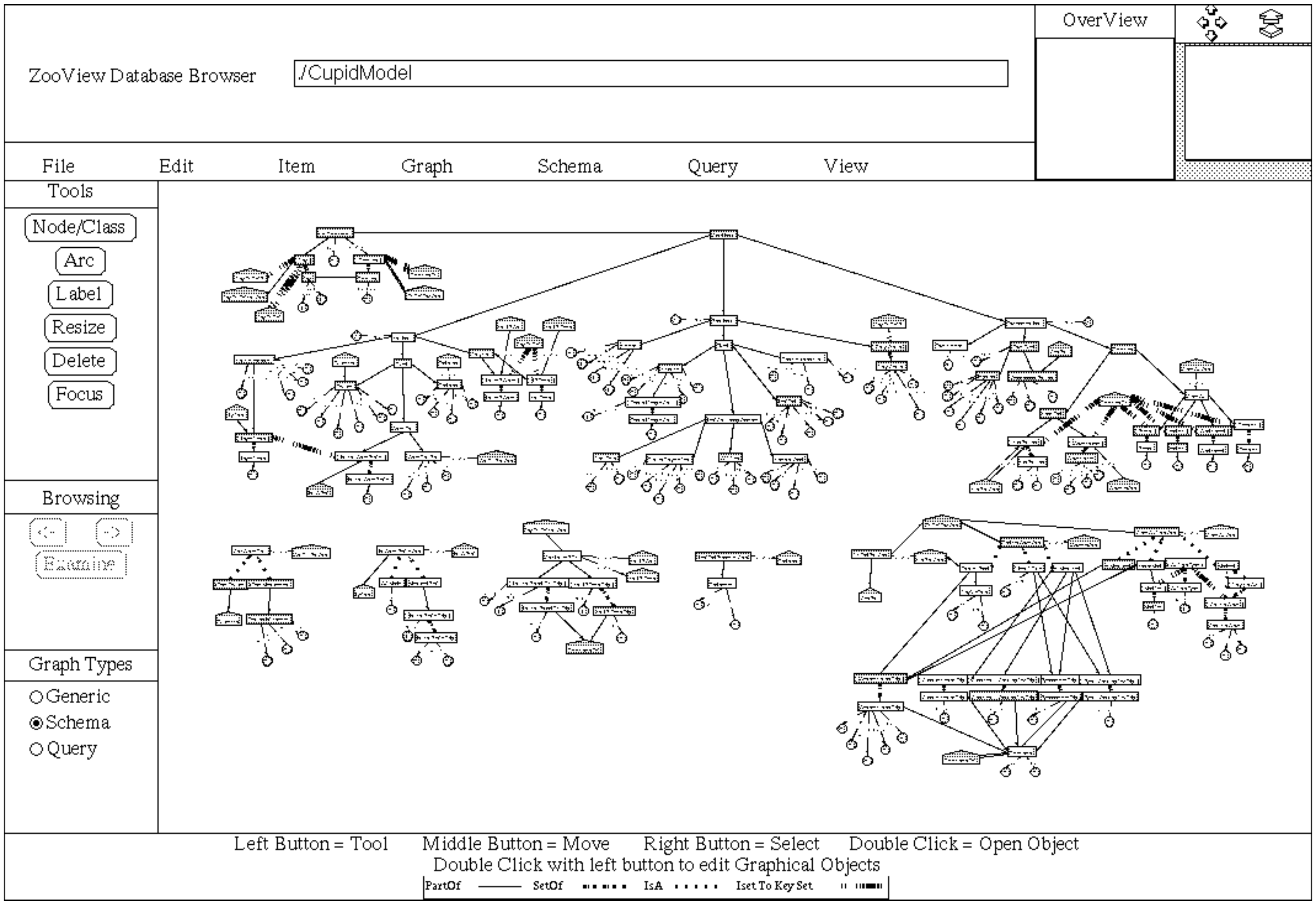

Figure 5: Moose schema for the Cupid input.

7. Summary

One of the major problems faced by experimental sciences today is the lack of adequate tools for the management of experiments and data. We have undertaken the effort to develop a desktop Experiment Management System that will provide adequate support for scientists involved in experimental studies. In this paper, we have identified some of the fundamental issues that must be addressed in designing such an EMS. We have developed an abstraction of the set of activities performed by scientists throughout the course of an experimental study, and based on that abstraction we have proposed an EMS architecture that can support all such activities. The proposed EMS architecture is centered around the extensive use of conceptual schemas, which express the structure of information in experimental studies. Schemas are called to play new roles that are not usually found in traditional database systems. We have provided a detailed exposition of these new roles and have described certain characteristics that the data model of the EMS must have in order for schemas expressed in it to successfully play these roles. Finally, we have presented the specifics of our own effort to develop an EMS, focusing on the main features of the data model of the system. The feedback that we have received from domain scientists that have used the data model to design their experiments have been encouraging and have strengthened our belief that our approach will be able to serve the needs of experiment management.

Our effort to develop an effective EMS is far from complete. There are several issues that we are currently investigating and many others on which we plan to work in the future. Those currently under way include architectural issues on how the User Interface and the Core DBMS should communicate, efficient support for initialization and follow-up requests, graphical representation of objects, storage structures and query optimization in the Core DBMS, and formal issues on data and query translation. Additional issues left for the future include work on the Experimentation Manager, developing an internal interface layer so that a variety of output analysis tools can be connected to the system, e.g., visualization tools, and developing a data-translator generator, which will be an easy-to-use toolkit for building data translators. In parallel to the above efforts, we will continue our collaborations with various domain scientists from different disciplines, so that the applicability of our findings to experiment management can be continuously tested and validated.

References

[AGS90] R. Agrawal, N. H. Gehani, and J. Srinivasan. OdeView: The graphical interface to ode. In *Proc. of the 1990 ACM-SIGMOD Conference on the Management of Data*, pages 34–43, Atlantic City, NJ, May 1990.

[ASL89] A. M. Alashqur, S. Y. W. Su, and H. Lam. OQL: A query language for manipulating object-oriented databases. In *Proc. 15th International VLDB Conference*, pages 433–442, Amsterdam, The Netherlands, August 1989.

[BCD89] F. Bancilhon, S. Cluet, and C. Delobel. A query language for the O2 object-oriented database system. In *Proc. 2nd International Workshop on Database Programming Languages*, pages 122–138, Salishan Lodge, CA, June 1989.

[BH86] D. Bryce and R. Hull. SNAP: A graphics-based schema manager. In *Proc. 2nd International Conference on Data Engineering*, Los Angeles, CA, February 1986.

[CDV88] M. J. Carey, D. J. DeWitt, and S. L. Vandenberg. A data model and query language for EXODUS. In *Proc. of the 1988 ACM-SIGMOD Conference on the Management of Data*, pages 413–423, Chicago, IL, June 1988.

[CM90] A. F. Cardenas and D. McLeod. *Research Foundations in Object-Oriented and Semantic Database Systems.* Prentice Hall, Englewood Cliffs, NJ, 1990.

[FJP90] 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.

[Fog84] D. Fogg. Lessons from a "Living In a Database" graphical query interface. In *Proc. of the 1984 ACM-SIGMOD Conference on the Management of Data*, pages 100–106, Boston, MA, June 1984.

[GGKZ85] K. J. Goldman, S. A. Goldman, P. C. Kanellakis, and S. B. Zdonik. ISIS: Interface for a semantic information system. In *Proc. of the 1985 ACM-SIGMOD Conference on the Management of Data*, pages 328–342, Austin, TX, May 1985.

[HL92] J. Hartmanis and H. Lin. *Computing the Future.* National Academy Press, Washington, DC, 1992.

[IL89] 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.

[ILH92] Y. Ioannidis, M. Livny, and E. Haber. Graphical user interfaces for the management of scientific experiments and data. *ACM-Sigmod Record*, 20(1):47–53, March 1992.

[KKS92] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proc. of the 1992 ACM-SIGMOD Conference on the Management of Data*, pages 383–392, San Diego, CA, June 1992.

[KL89] M. Kifer and G. Lausen. F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *Proc. of the 1989 ACM-SIGMOD Conference on the Management of Data*, pages 134–146, Portland, OR, June 1989.

[KM89] M. Kuntz and R. Melchert. Pasta-3's graphical query language: Direct manipulation, cooperative queries, full expressive power. In *Proc. 15th International VLDB Conference*, pages 97–105, Amsterdam, The Netherlands, August 1989.

[Liv87] M. Livny. DELAB - a simulation laboratory. In *Proc. of the 1987 Winter Simulation Conference*, Atlanta, GA, December 1987.

[MF91] V. M. Markowitz and W. Fang. SDT - a database schema design and translation tool. Technical Report LBL-27843, Lawrence Berkeley Laboratory, Berkeley, CA, May 1991.

[MS92] V. M. Markowitz and A. Shoshani. Query specification and translation tools. Technical Report LBL-31155, Lawrence Berkeley Laboratory, Berkeley, CA, April 1992.

[Nel90] D. Nelson. The laboratory notebook technical manual. Technical Report LA-UR 88-1256, Los Alamos National Laboratory, Los Alamos, NM, 1990.

[P^+92] J. Paredaens et al. An overview of GOOD. *ACM-SIGMOD Record*, 20(1):25–31, March 1992.

[RC87] T. R. Rogers and R. G. G. Cattell. Entity-relationship database user interfaces. In *Proc. of the 6th International Conference on ER Approach*, New York, NY, November 1987.

[SM91] E. Szeto and V. M. Markowitz. ERDRAW - a graphical schema specification tool. Technical Report LBL-PUB-3084, Lawrence Berkeley Laboratory, Berkeley, CA, May 1991.

[WK82] H. K. T. Wong and A I. Kuo. GUIDE: Graphical user interface for database exploration. In *Proc. 8th International VLDB Conference*, Mexico City, Mexico, September 1982.

[WTM^+92] J. Wiener, O. Tsatalos, R. Miller, M. Livny, and Y. Ioannidis. Direct modeling of context-dependent associations in semantic data models, November 1992. In preparation.

[Zei76] B. P. Zeigler. *Theory of Modeling and Simulation.* John Wiley & Sons, New York, N.Y., 1976.

[Zlo77] M. M. Zloof. Query-by-Example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.

[ZM89] S. B. Zdonik and D. Maier. *Readings in Object-Oriented Database Systems.* Morgan Kaufmann, San Mateo, CA, 1989.