The Concept and Design of a Virtual Laboratory

Lynn Mercer
Przemyslaw Prusinkiewicz
James Hanan
Department of Computer Science
University of Regina
Regina, Saskatchewan
Canada S4S 0A2

Abstract

In this paper we introduce the concept of a virtual laboratory as an electronic metaphor for a "real" lab. The virtual laboratory allows the user to discover knowledge through simulated experiments guided by hypertext documents. It contains all the elements of a good physical laboratory: a wide range of objects and tools which can be assembled into experiments, a reference book, and a notebook for recording thoughts, inspirations, results and conclusions. The concept is illustrated by an example of a laboratory design.

Keywords: microworld, hypertext, software environment for interactive visualization, object-oriented programming.

1 Introduction

Powerful graphics workstations offer unprecedented hardware capabilities for conducting simulated experiments using interactive visualization programs. However, the system software support for this type of application is inadequate. For example, the user may find himself overwhelmed by hundreds of data files representing different models and model components, some of which are appropriate as input for one simulation program, some for another. This paper presents the concept of a *virtual laboratory* which provides an environment for the management of simulated experiments.

A virtual laboratory, like its "real" counterpart, is a playground for experimentation. It comes with a set of objects pertinent to its scientific domain, tools which operate on these objects, a reference book, and a notebook. The reference book explains the concepts involved through text and illustrations, and serves as a guide to experiments. Once the concepts and tools are understood, the user can expand the laboratory by adding new objects, creating new experiments, preparing written reports and recording them in the notebook. An experienced user can further expand the laboratory by creating and installing new tools.

This paper presents the concept of a virtual laboratory and outlines a design implemented on Silicon Graphics workstations.

2 Virtual lab = hypertext + microworld

Technically, we define a virtual laboratory as a microworld which can be explored under the guidance of a hypertext system. The term "microworld" denotes an interactive environment for creating and conducting simulated experiments [16]. The motivation for exploring this microworld is based on the ability to see and create "neat phenomena" [9], provided in the form of graphical objects.

In a sense, both components of the virtual laboratory are described by Nelson in *Dream Machines* [11]. The pioneering role of this book in introducing the concept of hypertext is known, but under the heading *The Mind's Eye* the notion of a microworld is also anticipated:

> Suppose that you have a computer.
> What sorts of things would you do with it?
> Things that are imaginative and don't require
>   too much else.
> I am hinting at something.
> You could have it make pictures and show you
>   stuff
> and change what it shows depending on what
>   you do.

Since *Dream Machines*, many examples of microworlds have been described in the literature. Papert [12] refers to a LOGO microworld which can be used to discover the principles of turtle geometry. L.E.G.O. [4, 5] provides a user with an electronic ruler and compass, thus creating a microworld for Euclidean constructions. The Alternate Reality Kit [15] models a world in which objects have physical properties, such as velocity and mass, allowing the user to learn about Newtonian mechanics. A similar microworld domain is considered by Papert [12]. Thinglab [2] is defined as a kit for building microworlds for diverse applications; examples include experiments with geometric objects and with simulated electric circuits. Many other programs are available commercially or for research purposes. For example, the Game of Fractal Images [8] makes it possible to explore the mathematical microworld of Julia and Mandelbrot sets, while Compose [3] creates a microworld of dancing figures to assist in choreographic design.

All these microworlds provide little guidance to the user. This is consistent with Papert's concept of a microworld as a simulated environment for "learning without being taught", which he justified using Piaget's model of children as builders of their own knowledge [12]. However, this justification does not extend to comprehensive microworlds which provide an environment for a wide range of advanced experiments and cover a broad domain of science. They cannot be explored efficiently without guidance to the underlying concepts and the system itself.

This guidance could be provided in the form of a traditional book, but an electronic document is more suitable for integration with a microworld. One example is Mathematica [17] — a microworld for exploring mathematical phenomena. A user creates a Mathematica Notebook which may contain formulas, graphics and animation. The text and graphics are divided into cells and structured hierarchically, so the Notebook may contain several levels of detail. The user can scan the text at any level and may request an expansion of any portion of it. However, there are no links for jumping from one portion of text to another.

Plantworks [7] can be viewed as a simple virtual laboratory for developmental biology. Two components form its core: a program for simulating plant development called pfg, and a hypertext system called metatext [13]. Metatext provides the capability of nonlinear browsing through the database of available experiments. When an experiment is selected, it automatically invokes applications with the appropriate argument files and displays a corresponding textual description. By invoking pfg, the user can view the modeled plant as a static image or as an animated sequence illustrating the developmental process.

In contrast to most microworlds, Plantworks is an open system. It can incorporate most applications and data files available on the UNIX system and expand in many diverse directions. The open design is also one of the cornerstones of the virtual laboratory concept; in this sense the laboratory can be viewed as extending the functionality of UNIX.

3 Design of a virtual laboratory

Generally, a virtual laboratory may be divided into two components: the application programs, data files, and textual descriptions that comprise the *experiments*; and the underlying system support and utilities that provide the *framework* on which these domain-dependent experiments are built. Below we describe the design of the latter component, the domain-independent software, that has grown out of our experience with Plantworks.

3.1 The requirements

The following list specifies features considered essential to lab operation.

- *Consistent organization of the lab.* In the lab environment, experiments are run by applying tools to objects. An object consists of files that are grouped together so that they can be retrieved easily. The format of the objects must be sufficiently standardized to allow straightforward implementation of common operations such as object saving and deletion.
- *Inheritance of features.* It is often the case that several objects differ only in details. The mechanism of inheritance should be employed to manipulate and store such objects efficiently.
- *Version control.* Interaction with an object during experimentation may result in a temporary or permanent modification. In the latter case, the user should be able to decide whether the newly created object replaces the old one or should be stored as another version of the original object.
- *Interactive manipulation of objects.* The laboratory should provide a set of general-purpose tools for manipulating object parameters. For example, objects could be modified using control panels or by editing specific fields in a textual description of an experiment.
- *Flexibility in conducting experiments.* The user should be able to apply tools (programs) to objects (data files) in a dynamic way, while an experiment is being conducted. This can be contrasted to a static association established when the object is incorporated into the system.
- *Guidance through the laboratory.* The laboratory should include a hypertext system that imposes a logical organization on the set of objects, provides a textual description of the experiments, and makes it possible to browse through the experiments in many ways. Specific experiments should be invoked automatically when the corresponding text is selected, in order to facilitate demonstrations and assist a novice user.

The remainder of this section gives a detailed discussion of how these functions are supported in the current design of the laboratory.

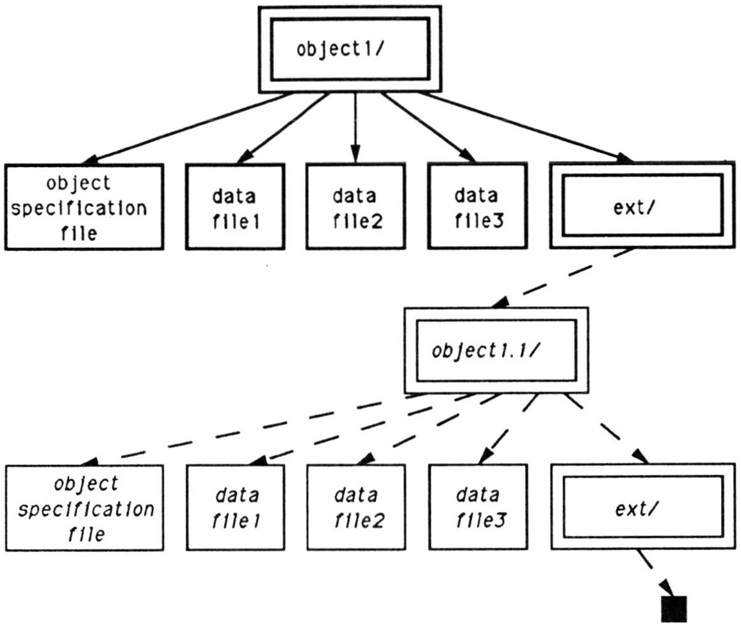

Figure 1: The hierarchical structure of objects.

3.2 Organization of the lab

The framework for the virtual laboratory consists of:

- a *file structure* for representing experimental units (objects);
- *utilities* for manipulating these objects; and
- a *hypertext system* that provides guidance through the lab.

So far, objects have been referred to in an intuitive way, relying on the analogy between a real and virtual lab. For example, if our interest is in the development of the moss *Phascum cuspidatum*, in a real laboratory we would experiment with a specimen of the plant, while in a virtual laboratory we explore the corresponding mathematical model. However, the analogy to real objects does not extend to the level of detailed object definition. Specific design decisions are needed for software development purposes.

In the current design, a laboratory object is defined as a directory containing two types of files and a subdirectory.

- The *data files* comprise our knowledge of a particular model.
- A *specification file* defines the data files which make up the object and the tools which apply to them.
- A directory of *extensions* lists objects which inherit some features of the current object.

The above definition is compatible with the UNIX file system. Consequently, the object-oriented file structure which provides the basis for lab operation can be represented by a hierarchy of UNIX directories and files (Figure 1).

3.3 Inheritance of features

The path of subdirectories leading to an object establishes the inheritance structure for the lab. Inheritance is based on the idea of specifying new objects in reference to objects which already exist [10]. The "old" object is called a *prototype* and the new one is its *extension*. The extension contains only those files which are different from the corresponding files in the prototype. Files that remain the same are *delegated* to the prototype by establishing links. In other words, the object directory will contain those files that are unique to the object, and links to files that are inherited from its prototype (Figure 2). This approach saves space, facilitates creation of objects similar to the prototype, and allows a single change in the prototype to propagate through all descendents.

3.4 Version control

To conduct an experiment, all files that make up the selected object are copied to a temporary location called the *lab table*. Consequently, manipulation of object parameters does not disturb the stored version. When the

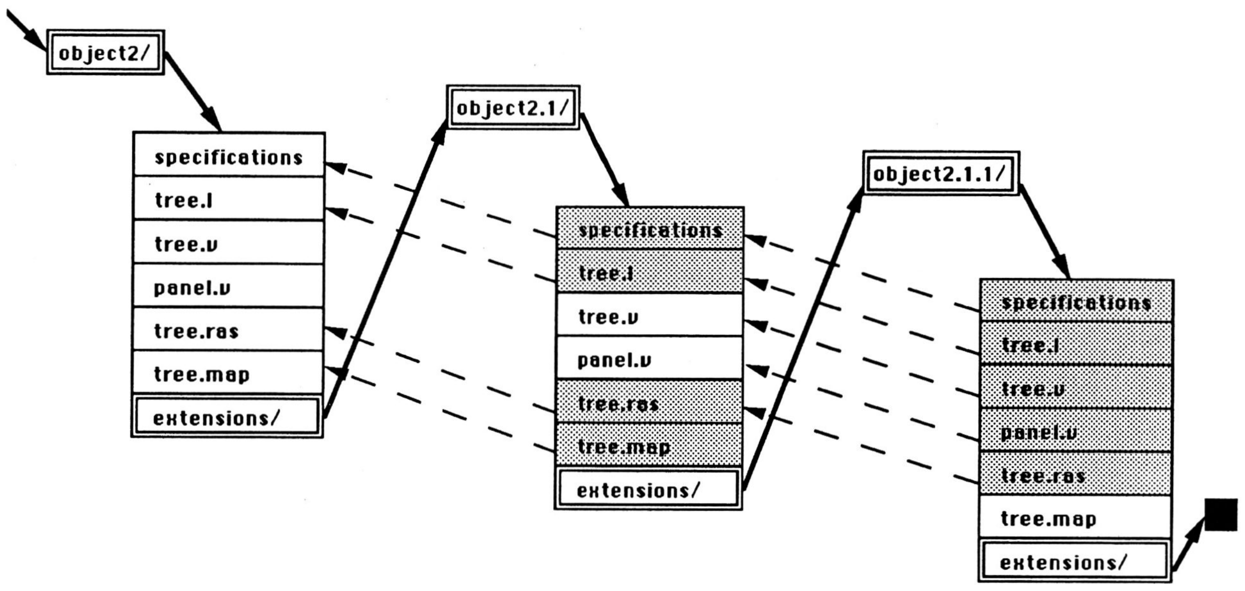

Figure 2: A prototype with a sequence of extensions. Shaded areas indicate links.

experiment is finished, the user may save the results by overwriting the original object or by creating an extension. In the latter case, the files on the lab table are compared with those in the prototype object; those files that differ from the prototype are saved, and links to the remaining files are established automatically.

3.5 Interactive modification of objects

The ability to easily manipulate the parameters in an experiment is an essential feature of the virtual laboratory. As a rule, all parameters involved in an experiment are supplied to the tools through the object's data files. In order to modify a parameter, the user edits the appropriate file, which is subsequently re-read by the application.

Though the editing of parameters can be accomplished using any text editor, in many cases parameter modification can be performed more conveniently using *virtual control panels* [14]. The current implementation of the laboratory provides the user with a general-purpose *control manager* which creates panels according to user-supplied configuration files. A configuration file specifies the appearance of each control in the panel and the format of the message to be sent as a result of control manipulation.

A control panel is typically used in conjunction with the editor *ed*. The editor receives messages from the panel through a UNIX pipe and special-purpose driver, and processes the data file accordingly. In other words, manipulation of a control on the panel causes a message to be sent to the editor, which interprets it as a command and edits the data file. This approach to parameter control has several advantages.

- It is consistent with the laboratory metaphor: we apply tools to objects, not to applications.
- Objects are kept up-to-date: all parameter modifications are recorded in the files which comprise the object.
- The design of applications is simplified. No special provisions for communicating with the panels are required; the ability to re-read input data files is all that is needed.
- The use of the editor as an interface between the panel and the application makes it possible to access and modify parameters in even the most complicated contexts.

The main limitation of the described approach is related to the one-way flow of information from the panel to the application. This makes it impossible for the application to update the controls when parameters are changed by the application itself. Also, unless special steps are taken, the initial values of the buttons and sliders reflect defaults specified in the configuration file rather than the actual values in the parameter file.

3.6 Dynamic application of tools to objects

Ideally, the user should be able to apply a tool to an object as a whole, without detailed knowledge of the programs involved or the component files. These effects are achieved through the object's specification file. It lists all files associated with an object, and the tools that can be applied to them. For example:

```
tree.l
tree.v
panel.v
tree.ras
*
image:
     generate:
          pfg tree.l tree.v
     snap:
          snap tree.ras
     paste:
          ipaste tree.ras

view:
     panel:
          display:
               panel panel.v | ped tree.v
          EDIT panel.v
     EDIT tree.v
```

Figure 3: A sample specification file.

The structure given in the specification file is used to create a hierarchy of menus associated with an icon representing the object (Figure 4). The end nodes in the hierarchy invoke tools that operate on the object. For example, selection of the item **image** followed by the item **generate** from the menus created using the above specification file invokes the plant modeling program **pfg**.

In order to avoid the repetitive specification of commonly used tools, they are described in a system-wide file, **tools**. For example, EDIT may be defined there as follows:

```
EDIT
edit:
     wsh -s 60,12 -c vi -w12
```

General-purpose utilities, which create a new version, save changes to an object, and perform other system functions, are also included in the object's menu. The program which produces the icon and handles menu selections is called the *object manager* and is invoked while browsing through the lab.

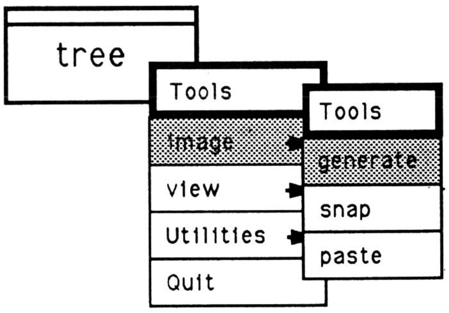

Figure 4: An object icon with menus.

3.7 Guidance through the lab

A user may browse through the objects in the lab either by following the hierarchy induced by the prototype–extension relation or by following hypertext links. The *object browser* provides rudimentary facilities for navigating through the hierarchy, moving down through successive extensions or up through previous levels. It appears on the screen as an icon bearing the name of the current object. At any time, the user may request that the browser invoke the object manager to place this object on the lab table.

It is often the case, however, that models separated in the prototype–extension hierarchy are related conceptually and should be associated for presentation purposes. Such associations are maintained by metatext — the UNIX-based hypertext system described in [13]. Like the object browser, metatext is represented on the screen by an icon with an associated menu. The menu items are specified in an *index* file which is read by metatext at the time of its invocation. Each menu item is associated with a *frame* file that specifies the object to be invoked and, optionally, the tools to be applied. These tools are described in terms of the object's menu hierarchy given in its specification file. For example, assuming the partial specification file presented in Figure 3, the tools to generate the image and display the control panel will be invoked automatically if options **image/generate** and **view/panel/display** are included as arguments when calling the object manager. Syntactically, the frame will contain the statement:

```
:object tree image/generate view/panel/display
```

Metatext can also be used to display a textual description of the object. The text is included in the frame and is formatted using *troff*. When a metatext menu item is selected, this text may be displayed along with the object. The set of textual descriptions and corresponding objects forms an interactive document which guides the user through the lab in a manner independent of its hierarchical organization. One such document consists of the *public* index files and frames that form the lab's reference book. In addition, an individual user may develop *private* metatext files to create his own notebook.

While these hypertext documents may be viewed as a means of relating disconnected objects and providing additional information and comments, descriptions that pertains directly to an object should be incorporated into the object itself. In the current implementation of the laboratory, an object may include *intelligent captions* that provide an explanation of experiment parameters as well as a method for manipulating them. The editor for managing these captions, called hvi, can operate in two modes. The *full edit* mode allows the user to create a caption consisting of text with embedded special fields. The *hypertext mode* enables the user to perform functions associated with the special fields while protecting the text fields from change. For example, some special fields can be edited in order to change parameter values. Other fields may act as buttons. Thus, an intelligent caption is similar to a control panel in that both may modify data files.

4 Conclusions

In this paper we have introduced the concept and presented an example of the design of a virtual laboratory. The objective of the laboratory is to provide an environment for conducting simulated experiments, similar to a microworld but capable of encompassing a broad range of concepts. The laboratory is an open system where a user may introduce new tools and create new experiments easily. It also provides a means to guide the user through the experiments and the underlying concepts. This latter function is fulfilled by a hypertext system: an interactive document that includes text and illustrations, and links them to the experiments. The extensibility of the laboratory is based on the UNIX philosophy of building upon existing utilities. The proposed object-oriented organization of data within the standard UNIX file system and the use of a hypertext system that supports UNIX shell commands, create an environment that supplements the existing system and becomes a general-purpose utility for conducting many diverse projects.

The virtual laboratory described here has been used as a software environment for experimenting with simulated plant models and fractals. In this application, it offers a number of benefits:

- The organization of data files into objects, complete with textual descriptions, alleviates the problem of finding files related to a particular experiment. This ease of access is particularly important in two situations: when the user wants to return to experiments performed in the past, and when experiments are to be run one after another for demonstration purposes.
- The virtual laboratory provides a good environment for cooperative work. A contributor of a new model creates the corresponding object and incorporates it into the laboratory. Other users can access the new object by selecting it from a menu associated with the object browser or metatext. In this way, the work of many people can be integrated into a coherent system.
- The concept of a lab table makes it possible to experiment freely with an object, and provides a simple method for saving the results. Combined with the inheritance mechanism, several versions of an experiment may be created easily and stored efficiently.
- The ease of changing parameters using virtual control panels prompts an extensive exploration of the models. On several occasions, the full potential of a model was not realized until a virtual control panel was applied to facilitate parameter modification.
- The virtual laboratory is an attractive research and learning environment.

The concept of designing the lab as an extension of UNIX has made it possible to develop it in a gradual way. Several extensions are still possible and some open problems remain.

- Although the version control mechanism facilitates the creation of new extensions, there is no laboratory support for creating new prototypes at the top of the hierarchy. At the present time they must be created by moving and copying the appropriate files using the UNIX shell.
- Object specifications should be extended to include information about the relationships between the components of an object. This information could be used to update related files, in a manner similar to that provided by the UNIX *make* facility.
- The object browser should provide a better visual interface for navigating through the object hierarchy. In the existing implementation, the icon representing the browser bears the name of the current object and its menu lists the object's extensions, but the position of the object within the entire hierarchy is not shown.
- Currently, there is no visual association between the object's icon and the windows it creates. This leads to confusion when several objects are on the lab table at one time. Possible solutions include using the object's name as the window title, connecting the related windows with lines as in ConMan [6], or allocating a specific portion of the screen to each object.

- When an object is removed from the table, all the associated windows should be closed automatically. This involves keeping a record of the processes invoked and signalling them when the object is removed. Although this would seem a simple task under UNIX, it is not easily achieved on IRIS workstations, where a process ID is changed as soon as the process opens a window.
- Metatext provides only basic hypertext functionality. In order to get a feel for improvements which may result from the use of a more sophisticated hypertext system, we are investigating the possibility of using HyperCard [1]. An attractive configuration for research purposes consists of a Macintosh II running HyperCard, connected to an IRIS workstation running simulation programs. This configuration offers the best of both worlds: an extensive hypertext environment and excellent interactive graphics capabilities.

Acknowledgements

Rob Morris brought the paper by Lieberman [10] to our attention, implemented the text editor *hvi*, and participated in many fruitful discussions. The comments of the anonymous reviewers were very helpful. The research reported in this paper was supported by grants from the Natural Sciences and Engineering Research Council of Canada and from Apple Computer, Inc. Facilities of the Department of Computer Science at the University of Regina were also essential. All support is gratefully acknowledged.

References

[1] B. Atkinson et al. HyperCard. Software for Macintosh microcomputers, 1987.

[2] A. Borning. The programming language aspects of ThingLab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, 3(4):353–387, 1981.

[3] T. Calvert et al. Compose. Software for Macintosh microcomputers developed at Simon Fraser University, Burnaby, Canada, 1989.

[4] N. Fuller and P. Prusinkiewicz. Applications of Euclidean constructions to computer graphics. *Visual Computer*, 5:53–67, 1989.

[5] N. Fuller, P. Prusinkiewicz, and G. Rambally. L.E.G.O. — an interactive system for teaching geometry. In *World Conference on Computers in Education.*, pages 359–364, Norfolk, Virginia, 1985.

[6] P. Haeberli. ConMan: A visual programming language for interactive graphics. *Computer Graphics*, 22(4):103–111, 1988.

[7] J. Hanan. Plantworks: A software system for realistic plant modeling. Master's thesis, University of Regina, Regina, Canada, 1988.

[8] H. Jurgens, H.-O. Peitgen, D. Saupe, and M. Permet. The game of fractal images. Software for Macintosh microcomputers developed at the University of Bremen, Bremen, West Germany., 1988.

[9] W. Lawler. Designing computer-based microworlds. In M. Yazdani, editor, *New Horizons in Educational Computing.* Ellis Horwood Ltd., London, 1984.

[10] H. Lieberman. Using prototypical objects to implement shared behavior in object oriented systems. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 214–223, New York, 1986. Association for Computing Machinery.

[11] T. H. Nelson. Computer Lib and Dream machines, 1980.

[12] S. Papert. *Mindstorms: Children, computers, and powerful ideas.* Basic Books, New York, 1980.

[13] P. Prusinkiewicz and J. Hanan. A hypertext environment for UNIX. In *Graphics Interface '88 Conference Proceedings*, pages 50–55. Canadian Information Processing Society, 1988.

[14] P. Prusinkiewicz and C. Knelsen. Virtual control panels. In *Graphics Interface '88 Conference Proceedings*, pages 185–191. Canadian Information Processing Society, 1988.

[15] R. Smith. The Alternate Reality Kit. In *1986 IEEE Computer Society Workshop on Visual Languages*, pages 99–106, Washington, D.C., 1986. IEEE Computer Society.

[16] P. Thompson. Mathematical microworlds and intelligent computer-assisted instruction. In G. Kearsley, editor, *Artificial intelligence and instruction: Application and methods.* Addison-Wesley, Reading, Massachusetts, 1987.

[17] S. Wolfram. *Mathematica: A system for doing mathematics by computer.* Addison-Wesley, Redwood City, California, 1988.