A Virtual Laboratory Notebook for Simulation Models

A.J. Winfield
Computing Laboratory, The University of Kent at Canterbury, Canterbury, Kent CT2 7NF, UK. ajw2@ukc.ac.uk

In this paper we describe how we have adopted the laboratory notebook as a metaphor for interacting with computer simulation models. This 'virtual' notebook stores the simulation output and meta-data (which is used to record the scientist's interactions with the simulation). The meta-data stored consists of annotations (equivalent to marginal notes in a laboratory notebook), a history tree and a log of user interactions. The history tree structure records when in 'simulation' time, and from what starting point in the tree changes are made to the parameters by the user. Typically these changes define a new run of the simulation model (which is represented as a new branch of the history tree). The tree shows the structure of the changes made to the simulation and the log is required to keep the order in which the changes occurred. Together they form a record which you would normally find in a laboratory notebook. The history tree is plotted in simulation parameter space. This shows the scientist's interactions with the simulation visually and allows direct manipulation of the parameter information presented, which in turn is used to control directly the state of the simulation. The interactions with the system are graphical and usually involve directly selecting or dragging data markers and other graphical control devices around in parameter space. If the graphical manipulators do not provide precise enough control then textual manipulation is still available which allows numerical values to be entered by hand. The Virtual Laboratory Notebook, by providing interesting interactions with the visual view of the history tree, provides a mechanism for giving the user complex and novel ways of interacting with biological computer simulation models.

1 Introduction

Simulations of the natural world are often used to conduct virtual experiments. In a real world experiment a scientist will make detailed records of an experiment as it is performed. These observations, notes and theories will be recorded in a laboratory notebook so that later it is possible to reconstruct what was done, when and why. In the virtual world of the computer simulation it is often the case that much of this detailed record keeping is ignored, potentially leading to important information being overlooked or lost.

Many simulations are not run on their own but are run inside an interactive steering environment. Interactive steering allows the output of a simulation to be viewed as it happens and then based on what is seen changes can be made to the state of the running simulation. Use of steering environments such as SCIrun[1] and CSE[2] tend to increase the amount of interesting data which is generated and thus makes the lack of detailed records an even more important problem to be solved.

We consider that running a simulation in an interactive steering environment is an experiment and, therefore, should be treated in similar ways to experiments in the real world. This suggests that we must store at least as much information as is kept for a real experiment and probably even more. If we continue with the notion of a simulation in a steering system being an experiment then other features of the real world should be taken into account. An example of such a feature is the fact that in reality all measurements are made to a finite accuracy. Although it is sometimes the case that we use a simulation to avoid this problem, it is still sensible to see if measuring accuracy would cause a major effect. This can be achieved by running user selected parameter values at each end of the possible range of values that any measuring error would allow.

The development of the virtual notebook has been motivated by the increasing use of simulations and steering systems in the biological sciences. These simulations are often complex with large numbers of variables, many of which may be inter-related. With large numbers of variables and unknown relationships between them the only sensible way of exploring the behaviour of biological simulations is to use some form of steering system. We chose to write our own steering environment, instead of using one of those already in existence, because it is our belief that the structure of the steering environment should promote good experimental practices and should support the production of usable results.

The basic aim of the virtual laboratory is to provide an interactive steering system that stores the data in a form which has all the structure and detail expected in a real laboratory notebook. It should also give an interactive user interface which allows direct graphical manipulation of views of the parameters of the simulation along with methods for simulating features of the real world such as the effects of limited measuring accuracy.

2 Motivation

We take as an example application of the virtual laboratory notebook (and the original motivation for developing it) the exploration of a spatially-explicit, individual-based ecological simulation model which has recently been developed in the Computing Laboratory at the University of Kent at Canterbury[3]. The model simulates the spread of the pathogen Barley Yellow Dwarf Virus through a cereal field.

Barley Yellow Dwarf Virus (BYDV) is an economically important pest of cereal crops and wild grasses worldwide[3]. It causes yellowing of the leaves and often results in significant yield loss in cereals. A simulation model was developed to allow investigation of the epidemiology of virus spread through a cereal field and the population dynamics of the aphid vector of the virus. The simulation model kept track of the position and state of the individual aphids and considered their effect on individual barley plants. Even a simple model produced prodigious amounts of data which necessitated the use of a scientific visualisation system such as NAG Explorer in order to display and interpret the results generated by the simulation model. Systematic investigation of the simulation model's parameters, such as sensitivity analysis required greater support than could be obtained from script files, log files and a conventional laboratory notebook. A convenient means of storing simulation results and relating these to the parameter values which had been used to generate them was also required. Finally, a simple mechanism for coupling existing computer simulations into a framework which provided for their exploration and visualisation was desirable.

Hence the specification for a 'virtual laboratory notebook' was conceived, a version of which is described in the remainder of this paper. It should be noted that while the original motivation for the work was the exploration of an ecological simulation model, there is no reason at all why the virtual laboratory notebook should not be applied to other simulation models in the biological sciences such as physiological or biochemical models. Or indeed it could be applied to simulation models developed in disciplines other than the biological sciences.

3 Previous Work

In previous work on implementing steering systems, much of the emphasis has been on the methods by which the system may be built. Examples of such work are SciRun[4,1] and CSE[2,5]. Although some work has been done to add logging of the state of variables in the CSE system[6] it is not intended to be used to log the complete state of a simulation. In our system we require that the complete state of a simulation is logged so that none of the information generated by the interactive steering environment is lost.

At present the systems which implement the most complete storage of data from the simulation and provide the most interesting interactions with it are GRASPARC[7] and HyperScribe[8]. These systems introduce the idea of logging everything about the state of the simulation at each point in time to allow restarting the simulation with altered parameters. They also introduce the data structure upon which our system relies heavily to structure the storage. The data structure is a history tree which is described in detail in section 3.1. It is the ideas that these systems present that we have extended in our work.

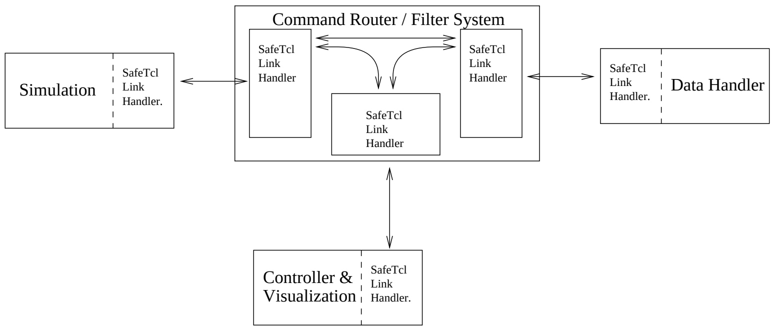


Figure 1: Block view of the system

3.1 History Trees

A history tree[7] is a historical record of the structure of changes made to the parameters of a simulation. During the exploration of behaviour of a simulation many runs may be required from different starting points to find phenomena of interest. In other words, the history tree documents and models the search process involved in exploring the simulation. Each branch in the tree is formed when a change is made to any parameter or set of parameters in the simulation. Therefore, each branch of the tree corresponds to a decision by the scientist.[7] Within each branch a sequence of snapshots of data is stored; these capture the entire state of a simulation. The 'entire state' is the information required by the simulation to be able to return to the current point in the run, with any other information which may be of interest to the scientist examining the output.

By using the tree structure the simulation may be returned to any point in the parameter space which has been visited by the simulation and then restarted with an altered state. This ability facilitates the search for interesting features in the output of the simulation without having to restart the simulation from the beginning.

4 System Overview

The overall system consists of a simple distributed system written using Tcl/Tk[9,10] to handle the low level networking. All information is transmitted using Tcl procedure calls sent between the processes which are then executed by the receiving end of the communications link inside a safe slave interpreter.

The use of a Safe-Tcl slave interpreter means that the connecting clients of the routing process can install extensions directly into the control process without too much risk to the process owners' account. As stated by Ousterhout *et al*[11], 'Safe-Tcl is a mechanism for controlling the execution of programs written in the Tcl scripting language. It allows untrusted scripts to be executed while preventing damage to the environment or leakage of private information.'

The system contains four major parts: simulations, the visualisation and control interface, the data storage handler and finally a central command router (see Figure 1). Each part contains a Tcl script to handle high level interactions in the system and an extension to the Tcl language to provide the extra functionality required of each system component. The parts of the system are separate processes which can be running on different machines.

5 Implementation

5.1 Interface to simulations

Simulations are interfaced to the system by altering them to present the required application program interface and then linking them with a library which implements the connection controls with the command router. The command router is the central element of the system which controls the transfer of data and commands between all the other parts of the system. By making the interface very simple it is hoped that alterations to existing simulations will be trivial. The library is actually a SWIG[12] generated wrapper which turns the simulation interface into Tcl commands. SWIG is a program which automates the construction of extensions to various scripting languages. These commands are then used by a simple Tcl script to create the desired behaviour.

5.2 Data storage

The data is stored using HDF[13] in a structure that allows (as far as possible) arbitrary types of parameters, although data structures such as trees must be flattened into an array form to be stored. The flattening of structured data is only required for the parameters to the simulation; other structural elements such as the history tree representation are stored without being converted to an array. 'The Hierarchical Data Format, or HDF, is a multi-object file format for sharing scientific data in a distributed environment.'[13]

HDF was chosen to store the data produced by the simulation was made because it is both portable and has the ability to store complex structures. The structures which HDF has can easily be utilised to store the history tree and other structural information, which comprises the content of the

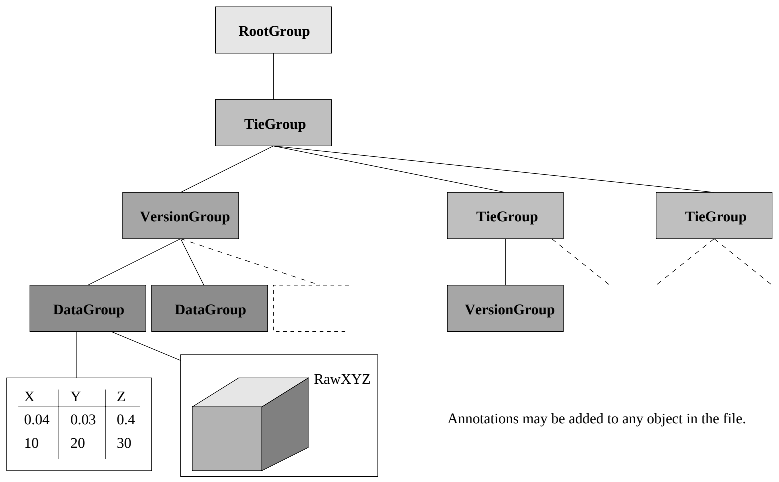


Figure 2: The Structure of the HDF datafile

virtual notebook. HDF is also designed to be very good at handling scientific data (multi-dimensional arrays) which is the form in which a very large amount of simulation data is created.

The structure of the file follows on from the structure of the history tree. The Root Group[a] is a link to individual runs of the simulation. That is a complete restart of the simulation with everything reset to the initial value except for the random number generator seeds. The history tree for each run is represented by the Tie Groups and the branches of the tree are held in the Version Groups. The data itself is held in Data Groups, each containing the full state of the simulation at a given timestamp. For a view of the file format used to store the history tree, see Figure 2 which shows the structure graphically.

In order to provide access to the data stored in the HDF file in a simple way we have written a C++ library to control the structure. A Tcl wrapper which uses SWIG was written to make the file accessible from the Tcl scripts which are used to create the simple distributed system in Figure 1.

[a] A group is a stored as an HDF Vgroup which is an array of references to other items in the file.

5.3 Data and Command Router

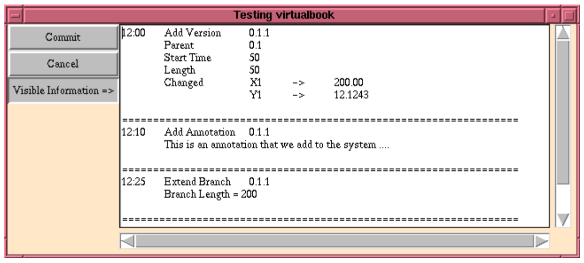
The central router (Figure 1) exists to simplify connection with the other parts of the system by giving a single point of access with a well known interface. It reduces the number of commands required by the clients to transmit data to groups of other clients. Most importantly, clients can add their own code to the routing process to make custom data filters to reduce unnecessary network traffic between the parts. Due to external processes installing their own code into the command router the basic functionality of the process is very simple: the command router has to register a client, route a command, register a command filter and execute a command[b]. Everything else is added by the clients as they initialize their connection to the command router.

5.4 The User Interface

The user interface is a visual front end onto the rest of the system. It provides a variety of methods by which the stored data may be viewed and manipulated. All of the views are constructed using Tcl/Tk and a Tcl extension VTK[14] which provides a large graphical visualisation library. The views may be grouped together to allow the user to interact with as many parameters as they require. Changes within a group are added together and then sent to the simulation as a single request message when the user decides that they have made all the required changes. The changes are performed by a message being sent to the simulation. This message contains all the required information to set the state of the parameters of the simulation to the correct new state and a single integer to say how many time steps for which the simulation should generate data.

It is possible to present many views of the history tree to the user for interaction purposes. The basic views are a notebook view, a history tree view and a parameter view. The notebook view (Figure 3) summarises the annotations which the investigator might make when experimenting with the simulation, interactions with the simulation and possibly some data. The interactions found in the graphical views can be performed in the notebook view; it is just harder to visualise what the changes to the simulation would look like. The only advantages to the notebook view are that it gives access to the Tcl interpreter, which allows scripts to be written to perform complex alterations to a simulation's state, and it gives a summary of all the annotations which are invisible in the purely graphical views.

[b]This executes the command in the scope of this connection's own safe interpreter



```
12:00   Add Version     0.1.1
        Parent          0.1
        Start Time      50
        Length          50
        Changed         X1      ->      200.00
                        Y1      ->      12.1243

=================================================================
12:10   Add Annotation  0.1.1
        This is an annotation that we add to the system ....

=================================================================
12:25   Extend Branch   0.1.1
        Branch Length = 200

=================================================================
```

Figure 3: The Notebook View

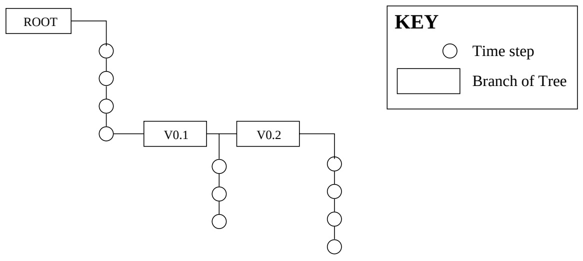


Figure 4: The History Tree View

The rest of the views are graphical and all show the history tree in some way. The most basic graphical view is the pure history tree view (see Figure 4). This view is similar to the view presented in GRASPARC[7], and has similar manipulation abilities. These include: adding new versions, viewing data at a point (a snapshot of the simulation's state), requesting that more data be added to a currently available branch of the tree and annotating items in the datafile. The tree view is also used to allow differentiation between coincident points where these occur in the parameter view. A final use for the history tree view is to work like a directory viewer, hiding and revealing whole sub-branches from all views of the data.

The final, and most interesting view, is the parameter view. This is a three dimensional plot of the history tree (see Figure 5) using three user selected simulation parameters to position the points. The operations on this view are much the same as on the others but due to the fact that we have the values in a visible graph the operations are performed by directly manipulating the points and lines shown on the graph. The idea of drawing the history tree in positions that suggest changes in the value of parameters that the nodes represent is something in the HyperScribe system. The difference between HyperScribe and our system is that in HyperScribe the positioning of points is up to the user, whereas in our system points are located according to the parameter values they represent.

There are many different interactions available within the parameter view. The interactions can be placed into two groups: single point and multi-point operations. With single point operations information may be requested about the selected point or that point maybe moved to create a new branch in the history tree. Multi-point operations also allow information to be retrieved about the selected points. In this case viewing the data may require that a sequence of information be output for visualisation, the rate at which this happens is user controlled (up to a limit which is governed by the speed of the visualisation). With fast enough visualisation output from such a sequence of data points, what is displayed will appear to be an animation. Modifications with multi-point operations repeat the existing user changes within the parameters of the selected points, adding on the new change made by the user. When adding the new change it may be applied relative to the new values of the parameters, relative to the old values of the parameters or fixed to the value the user selected. Added to the operations is another ability to automate a set of changes by selecting the top, bottom and step-size of a range of individual alterations.

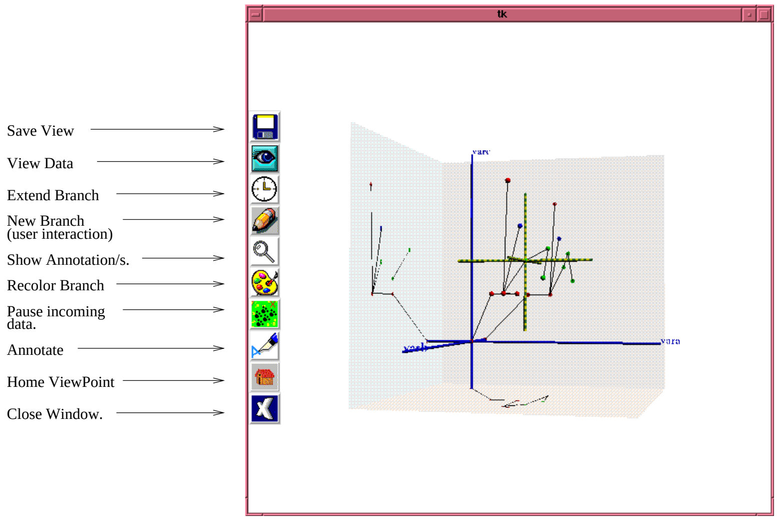


Figure 5: The Parameter Interactor

6 Using the virtual laboratory notebook

In order to use the virtual laboratory notebook, some changes to a simulation are required before it can be used by the steering system. The first change is to add a few information functions, these include: `simname`, which gives the symbolic name by which this simulation will be known when it is part of the system; `initialState`, which sets all external variables to their initial values, and `miscinfo` which is just used for information purposes (it appears in the user interface instead of the symbolic name). Along with the informational functions, other required functions include: `Initialise`, which initialises anything that might be required by the system as opposed to the simulation, and `Shutdown`, which should set the simulation to such a state that the process can exit. The final set of changes are a little more involved, but basically require turning the normally loop driven core of the simulation into one which is demand driven.

Converting the main loop of the simulation is actually a simple process. It involves creating three functions which embody the normal items found in the main loop of most simulations. `SetState` is executed before a time step is generated; its purpose is to get the parameter states from the Tcl interface so that it can set the current state of the simulation. `RunOne` takes the current state and does one complete time step's worth of calculations of the simulation. Finally, the `GetState` function exports the current simulation state to the Tcl interface. Once all the above functions have been written the normal program entry point (`main` in C and C++) has to be removed. This is because the library provides an entry point of its own.

7 Future Work

At present the system is mostly programmed using Tcl scripts. The use of a scripting language allowed us to get the system to work quickly and can be useful for writing extensions, but the speed of some parts of the system is unfortunately affected by the use of scripts. To improve the speed, by far the easiest option is to move as much of the code into C libraries which are then used to extend the Tcl language. The most pressing need is for VTK to have a data-flow editor to avoid coding visualisation pipelines. Without this it may be better in the long term to change the visualisation system to use a commercial product such as NAG Explorer which does have a data-flow editor.

We would also like to demonstrate the virtual laboratory notebook with a range of simulation models, not just ecological simulation models. In particular, we would like to confirm our contention that converting existing simulation models to run under the virtual laboratory notebook is a straightforward process, providing simulation model builders with a versatile tool for exploring the virtual worlds they have created.

Acknowledgements

Thanks must go to the Computing Laboratory at the University of Kent at Canterbury for giving me an EBS bursary without which I could not afford to do this research. Also thanks to D.R.Morse whose many many comments on this paper have made it readable.

References

1. Steven G. Parker, David Beazley, and Christopher R. Johnson. Computational steering software systems and strategies. Technical report, Department of Computer Science, University of Utah, University of Utah, Salt Lake City, UT 84112, July 1996. URL=http://www.cs.utah.edu/˜ sci/.
2. Robert van Liere and Jurriaan D. Mulder. CSE : A modular architecture for computational steering. Technical Report CS-R9615, CWI, 1996.

URL = http://www.cwi.nl/ftp/CWIreports/IS/CS-R9612.ps.gz.

3. Tim Hopkins and David R. Morse. The implementation and visualisation of a large spatial individual-based model using Fortran 90. Technical Report 18-96, University of Kent at Canterbury, Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent CT2 7NF, UK, October 1996. URL=http://www.cs.ukc.ac.uk/pubs/1996/42/index.html.
4. S.G. Parker, D.M. Weinstein, and C.R.Johnson. The SCIRun computational steering software system. In E. Arge, Brauset A.M, and H.P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 5–44. Birkhauser Press., Sep 1997.
5. Jurriaan D. Mulder and Jarke J. van Wijk. 3D computational steering with parametrized geometric objects. In *Proceedings of IEEE visualization 1995*, pages 304–311, 1995.
6. Jurriaan D. Mulder and Jarke J. van Wijk. Logging in a computational steering environment. Technical Report CS-R9613.ps.gz, CWI, 1995. URL=http://www.cwi.nl/ftp/CWIreports/IS/CS-R9613.ps.gz.
7. Ken Brodlie, Andrew Poon, Helen Wright, Lesley Brankin, Greg Banecki, and Alan Gay. GRASPARC - a problem solving environment integrating computation and visualization. In *Proc. IEEE Visualization1993*, pages 102–109. IEEE Press., 1993.
8. H. Wright and J.P.R.B Walton. Hyperscribe : A data management facility for the dataflow visualization pipeline. Technical report, NAG Ltd., February 1996. URL=http://www.nag.co.uk/0h/doc/TechRep/ietr.html.
9. John K. Ousterhout. *Tcl and the Tk toolkit.* Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass., Wokingham, Addison-Wesley, 1994, 1994.
10. Brent Welch. *Practical Programming in Tcl and Tk.* Prentice Hall, 1995.
11. John K. Ousterhout, Levey Y. Jacob, and Brent B. Welch. The Safe-Tcl security model. Technical report, Sun Microsystems Laboratories, 1997. URL=http://www.sunlabs.com/˜ ouster/safeTcl.html.
12. David M. Beazley. *SWIG (Simplified Wrapper and Interface Generator).* University of Utah, Salt Lake City, Utah 84112, version 1.0 edition, August 1996.
13. NCSA HDF Development Group. *HDF Users Guide v4.1r1.*
14. W. Schroeder, K. Martin, and B. Lorenson. *The Visualization Toolkit, an Object-Oriented Approach to 3D Graphics.* Prentice Hall, 1996.