Computational Experiments using Distributed Tools in a Web-based Electronic Notebook Environment

Allen D. Malony, Jenifer L. Skidmore, and Matthew J. Sottile

Department of Computer and Information Science, University of Oregon,
Eugene, Oregon 97403-1202

Presenting author: Allen D. Malony
Corresponding author: Allen D. Malony, malony@cs.uoregon.edu

Computational Experiments using Distributed Tools in a Web-based Electronic Notebook Environment

Allen D. Malony and Jenifer L. Skidmore and Matthew J. Sottile

Department of Computer and Information Science, University of Oregon, Eugene, Oregon 97403-1202

Abstract. Computational environments used by scientists should provide high-level support for scientific processes that involve the integrated and systematic use of familiar abstractions from a laboratory setting, including notebooks, instruments, experiments, and analysis tools. However, doing so while hiding the complexities of the underlying computational platform is a challenge. ViNE is a web-based electronic notebook that implements a high-level interface for applying computational tools in scientific experiments in a location- and platform-independent manner. Using ViNE, a scientist can specify data and tools, and construct experiments that apply them in well-defined procedures. ViNE's implementation of the experiment abstraction offers the scientist easy-to-understand framework for building scientific processes. This paper discusses how ViNE implements computational experiments in distributed, heterogeneous computing environments.

1 Introduction

The increasing application of high-performance computing, storage, and graphics systems to scientific problem solving coupled with the ever-widening access to and interaction with scientific tools and knowledge made possible by high-performance networking, offers the potential to remove physical constraints of space and time in scientific investigation. However, scientists routinely employ *processes* in their work that specify many aspects of experimental procedures, including measurement protocols, sequenced data analyses, process monitoring, and results presentation. These processes provide the framework for scientific study and are commonly reflected in the physical operations of the traditional laboratory environment. Scientists also follow processes in computational environments that include the use of simulation programs, databases, analysis and visualization tools, networked computing resources, and web-based documentation [2, 3]. However, whereas these computational "instruments" are highly advanced, the support for the computational science processes is often simplistic and limited. One reason is that the processes are naturally defined with respect to the scientific domain of abstraction and these abstractions are not easily mapped to the system-level complexities of the computational environment and infrastructure. In this paper, we discuss a web-based framework called *ViNE* designed to support computational scientific processes in a high-performance distributed environment.

The Virtual Notebook Environment (*ViNE*) is a platform-independent system that manages a range of scientific activities across distributed, heterogeneous computing platforms [10]. On the one hand, ViNE provides the web-based equivalent of common paper-based lab notebooks with additional features for notebook sharing, security, and collaboration. Other electronic notebook systems offer similar capabilities [4, 5, 6, 8] in addition to features such as automatic data acquisition [9] and image annotation [5]. On the other hand, ViNE is unique in its ability to represent, manage, and execute *computational experiments*; i.e., support scientific processes that involve the use of data, tools, and programs throughout the scientist's computing repetoire. ViNE hides system-level complexities, allowing scientists to specify and launch computational experiments using a visual specification language. The scientist is freed from concerns about inter-tool connectivity, data distribution, data management, and machine idiosyncracies. Experiment specification is abstracted in the notebook interface and results from experiments can be dynamically linked back into the notebook content.

We focus here specifically on ViNE's support for computational experiments. In particular, we describe how ViNE represents experimental processes in the notebook and executes the experiments in a distributed, heterogeneous system. In section §2, we give a high-level view of ViNE's system architecture. Section §3 briefly describes some of the components that make up the ViNE system. The framework for computational experiments is discussed in detail in Section §4. We conclude the paper with our future plans.

2 ViNE Architecture

The Virtual Notebook Environment provides functionality through five components: *browse*, *administration*, *data organization*, *tool organization*, and *experiment* [10]. The administration and browse components provide the basic notebook functions which include managing notebook structure (pages, chapters, content), security parameters, and navigation. Extended functionality such as describing data, tools, and analysis tasks are provided by the data organization, tool organization and experiment functional components, respectively. These five components work together to access, store and manipulate notebooks, data, and tools on a given machine.

ViNE is constructed to operate in a distributed, heterogeneous system, and its components are spread across the machines used in the computational environment. A *leaf* is the abstract entity that contains the functional components, notebooks, data, and tools. In addition, a leaf contains a communication server, called a *stem*, and a webserver. The stem is responsible for sending and receiving URL formatted messages that encode ViNE's commands and protocols between the leaves. The webserver provides the notebook's user interface.

A leaf can contain all five components, providing full ViNE functionality for the user. It is also possible to have a more restricted leaf where only the notebook administration and browsing components are available. This type of leaf is a notebook server that only provides the basic notebook functions and does not have any extended functionality. In addition, there are special leaves that do not contain any components, but do have stems and web servers. These leaves can be used to provide access to data stored in large data repositories or act as computation servers running specialized tasks, offering users with a wider spectrum of tools. These leaves can be thought of as data and computation servers that do not house notebooks. The leaves can be configured and located in a way that best fits the distributed system.

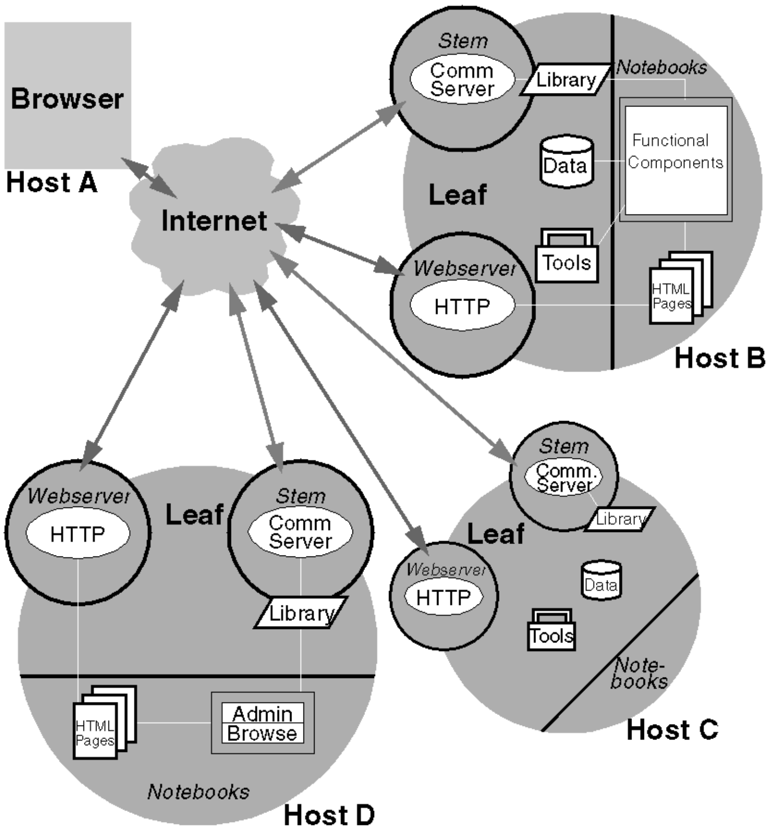

Fig. 1. Virtual Notebook Environment

Together, the leaves distributed across the computational platforms define the notebook environment. An example of a notebook environment can be seen in Figure 1. It contains a leaf that provides complete functionality (Host B), a computational and data server (Host C), and a notebook server (Host D). ViNE's architecture consists of one or more notebooks environments.

3 ViNE Components

The *data organization*, *tool organization*, and *experiment* components work together to provide the extended functionality of the notebook. The data and tool organization components implement the management facilities for a notebook's resources. Once data and tools have been described in the notebook, the user is able to combine their data and tools to perform analysis tasks, which we call *experiments*; see section §4.

The data component allows notebooks to reference data throughout the ViNE system. ViNE currently recognizes just two types of data, tabular and general data. Tabular data is a matrix of values, fully described by its dimensions. Both tabular and general data are treated as a flat files. A general data file is described by giving it a name, providing the name of the host it is located on, and the complete path to the file. Data that can be interpreted and understood when viewed, such as matrices and images, can be displayed on the notebook pages. All described data is available for use in the experiment control component.

The tool organization component maintains descriptions of available tools and their wrappers. Tools are categorized as either general or custom depending on whether they can be fully controlled from the command line (general) or require more complex interaction and control (custom). Wrappers are Common Gateway Interface (CGI) scripts that run a tool with the given parameters and can be accessed by a leaf's webserver. ViNE provides wrappers for general tools. To register a general tool with ViNE, the user enters its name, location, and a description of its input and output parameters. For a a custom tool, he/she must also supply a wrapper to interface between ViNE and the tool. The wrapper is responsible for accepting a complex string containing the input and output information, translating it for the specific tool, and executing the tool correctly. In either case, once the tool is registered, it can be used in experiments.

4 Computational Experiments

The ViNE experiment component provides users with the ability to create and execute distributed tasks for data analysis and manipulation. This component was designed with the target user in mind and provides a layer of abstraction between the system and user. This layer hides details related to the function and implementation of the underlying data transport and control mechanisms, leaving the user to concentrate only on the task they wish to accomplish. The experiment component is divided into two sub-components that together provide all necessary functions - the *experiment builder* and the *experiment controller*.

Before discussing the software architecture, it will be necessary to introduce the representation of experiments within the environment. In general, an *experiment* within ViNE is a sequence of data transfers and operations. An experiment is represented as a directed acyclic graph of computational nodes and data-flow connections. This representation is advantageous to the implementation for many reasons. For constructing experiments, one can easily construct the data-flow graph within a visual application. Well known algorithms for proving certain characteristics about graphs, such as cycle elimination, allow for simple error-detection at construction time to avoid potential problems at execution. Finally, the use of a graph representation in the controller provides the capability to easily describe parallel task execution within a single experiment.

One of the important design considerations within experiments was defining the input/output characteristics of the nodes and data-flow connections. As stated above, when a tool is introduced into ViNE the user must describe the I/O characteristics of the wrapper. Since experiments also involve data files and previously constructed experiments, the I/O treatment of such entities also must be defined. It is assumed that a data files is available for either reading or writing at any given moment. In order to protect data integrity, data files are allowed a single input during an experiment. Since read operations cannot damage data and data is assumed to be static once created, an arbitrary number of entities can receive input from a given data file. The I/O characteristics of existing experiments are defined so that any experiment may be used within a new experiment. More explanation of experiment construction and execution is necessary to describe this fully and is given in the experiment control section below.

4.1 Experiment Builder

Construction of experiments is accomplished using a Java [1] applet executed within any Java-enabled web browser. The builder provides a visual environment in which users may choose data, tools, and experiments contained within their notebook to create new experiments. Entities are chosen and placed into a canvas on which the user creates connections between the entities to represent data-flow within the experiment. Tools within an experiment may be configured within the applet by entering information into a configuration frame which appears when the tools is added or selected. These configuration frames use information from the tool description component to allow the user to set parameters for the tool execution. Tools which do not utilize the general wrapper architecture may provide a custom frame which is dynamically loaded into the applet to allow user configuration of the custom wrapper.

The custom tool configuration frames and other portions of the builder rely on its object-oriented design and implementation. The representation of an experiment within the builder is as a set of objects representing each node and connection. The node classes all derive from a generic parent containing the basic information shared by all node types, while providing an abstract API which each specific sub-class must provide regardless of the implementation. By using an approach such as this, the generic API provided to entities allows classes such as the configuration frames to be rapidly created and integrated into the basic system with no modification.

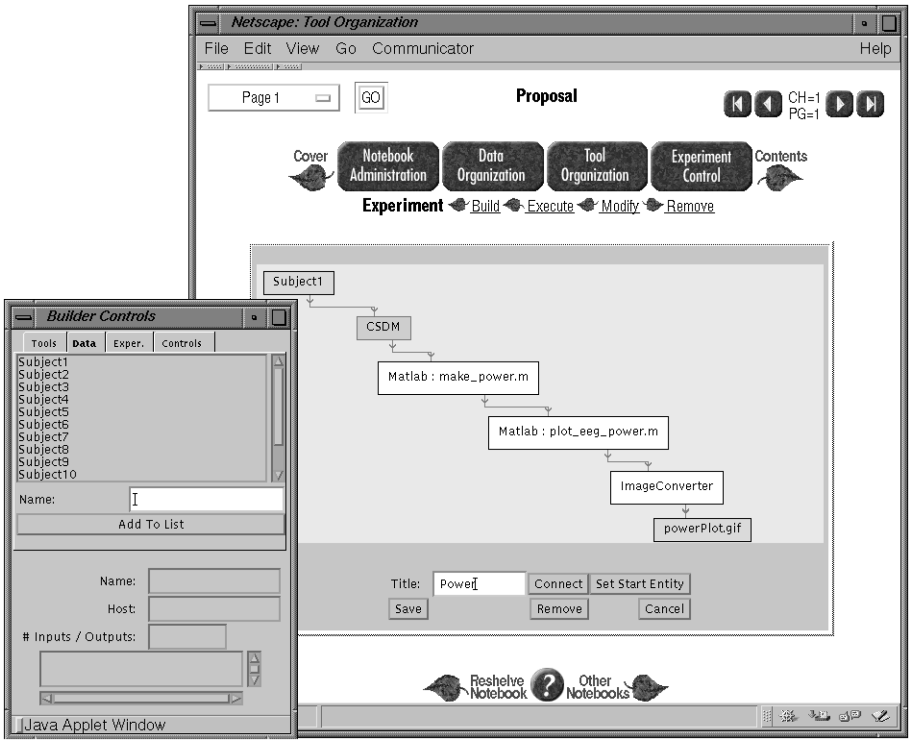

Fig. 2. Experiment Builder

In addition to the classes which represent the experiment itself, the experiment builder contains classes which perform experiment verification and routine GUI tasks. The GUI classes do not deserve much attention and simply provide a flexible canvas for drawing experiments while providing access to configuration information. The verification classes are able to rapidly test the experiment during construction to ensure that the resulting graph conforms to certain basic requirements. With the addition of data-flow connections, the verification class notifies the user and prevents the creation of dangerous cycles or impossible input/output combinations (such as an inappropriate number of input or output streams.) Database access scripts, implemented using Perl, provide a means for querying and modifying the database's existing experiment, tool, and data information. A simple CGI-based communication scheme to these backend database access scripts, in combination with the applet components, provide all required functionality for experiment creation.

4.2 Experiment Controller

Like the experiment builder, the ViNE *experiment controller* application is also implemented using Java for a variety of reasons. An object based design was used to implement the controller, taking advantage of Java thread, networking, and I/O constructs available within the default library. The controller utilizes the Java-based stems to perform actions such as locating data and tools, retrieving security information, and transferring data between leaves. At execution time the controller is invoked on a single leaf and manages data and tool invocation for the duration of the experiment.

Before execution the controller reads the description of the data flow graph as constructed in the builder applet. Since this graph was checked for errors during construction, this process is not required within the controller. The data flow graph is reconstructed as a network of objects within the controller, with each object containing required dependency information so that the correct order of operations will occur. Synchronization is accomplished by implementing each node as a thread object and using built in thread methods for monitoring dependency states. Reconstructing the experiment graph involves two steps - first, each node is created with all information relevant to finding and executing or moving the appropriate entity. If the node represents an existing experiment, the representative node is substituted by the entire experiment graph.

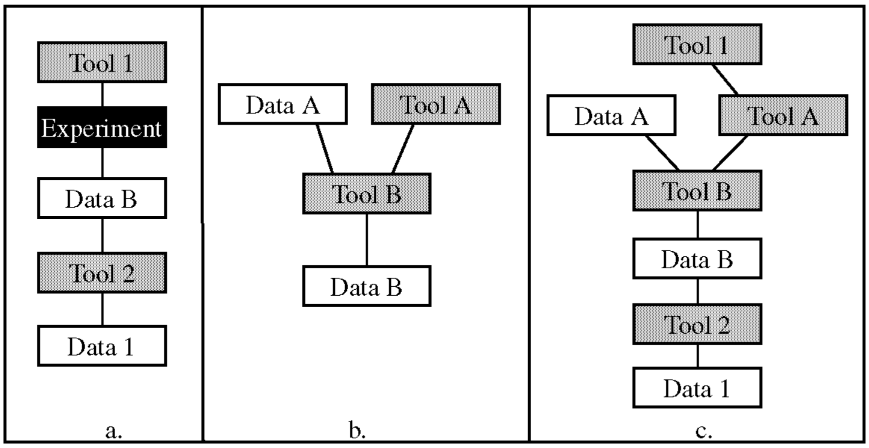

Fig. 3. Experiment Substitution : Experiment (a) contains inlined experiment (b), resulting in the actual executed experiment (c).

Once the nodes exist as threads, the connection information is read and each connection is established. Connections are established by giving each node with an input a reference to the node providing the input data. Nodes which are referenced as dependencies also are given a count of nodes who require their output. This is used later in managing execution workspaces. This node is then monitored by the receiving node in so that execution begins only when the output data is ready. Once all nodes and connections are created, all nodes are given a reference to a single semaphore thread and each thread is started. Once all threads are started, each attempts to **join** the semaphore which ceases execution when the last node is started. This causes all threads to start simultaneously. Each thread then attempts to **join** all threads it requires as input. Threads which have no dependency list start immediately while all others enter a waiting state. The experiment is completed when all node threads have terminated.

Experiment substitution for inlined experiments, as mentioned above, involves a more complicated I/O treatment than data or tools. Input to an inlined experiment depends on which nodes within the inlined experiment have no dependencies. If a tool occurs in this set of nodes with no dependencies, any input given to the experiment is passed to these tools regardless of whether or not it is used. If data files occur as nodes without dependencies, then incoming data is ignored for these particular nodes. Output from an inlined experiment is somewhat ambiguous to define. Since experiments are not required to have single input/single output behavior, an experiment may produce many data files as the final step of execution. Deciding which of these to pass along to the following nodes in the new experiment is difficult, and for obvious reasons such as performance and space constraints, all of the output data cannot simply be given as input leaving the receiver to decide which to use. For these reasons, output from an inlined experiment is assumed to arrive in data files defined within the inlined experiment, which can be utilized in the new experiment by explicitly adding the required data files as nodes.

Actual execution of a node is a simple process which occurs in the body of the thread objects. The first step in the lifetime of a thread is to wait for its dependencies to complete execution. Once this occurs, the thread queries each dependency thread for a list of available output data. Given this list of data files, the node creates an instance of a Java-based client to communicate with ViNEs Java servers. Using this client, the node checks what type of node it is to determine the next step. If the node is a data file, input from the previous node is moved into the notebook and a database entry is created for the new data if necessary (existing data of a given logical name is overwritten). At this point, the node terminates execution and threads that depend on it automatically start. If the node is a tool object, the execution process is somewhat more complex. The first step is to use the Java client to create a workspace on the host containing the tool, into which all input and output data is temporarily stored. The Java client is then given the set of inputs which are moved from the dependency nodes to the current workspace. Once all data is moved, the dependency nodes are notified and are allowed to remove their workspaces if all nodes which depend upon them have sent notification.

When all input data is within the workspace, the node then copies any additional files required for tool execution to the workspace. This is required for tools such as MatLab [7], in which programs may require additional modules to exist within the workspace where the execution occurs. As soon as all data

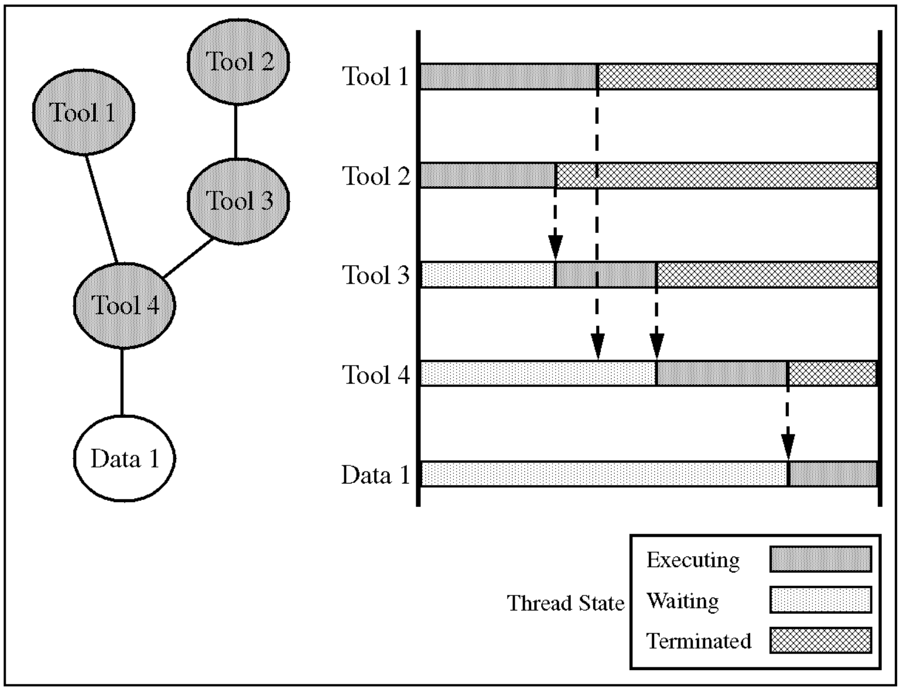

Fig. 4. Thread Lifetime

and other required files are completely moved, the client is used to invoke the CGI-based tool wrapper. When the tool has completed execution, the wrapper returns output information to the client, which then passes it to the node thread for further processing. The node uses this output information, along with a directory listing of the workspace to build the list of output files created during runtime. This list is then made available for other nodes which depend on this node, and execution of the thread terminates.

Experiment output is placed within the notebook of the user for future use and analysis. If, for example, a visualization is created, the user may then place this within a notebook page for later browsing. Status information of experiments is also stored within the notebook so users can monitor running experiments or analyze the behavior of previously executed experiments. Since experiments are run within the experiment controller, an arbitrary number of simultaneous experiments may execute, constrained only by system resources. Each executing experiment involves starting a new instance of the controller software, while all using a common Java server. By using a common Java server to which all clients connect for data movement and workspace creation, collisions of workspaces and intermediate file naming is prevented.

5 Future Directions

The current version of ViNE is a prototype being tested by scientists at the University of Oregon [2]. During this testing, we have identified a number of enhancements for future versions of the system. The notebook administration component will be extended with a framework for integrating editors appropriate to a variety of data content, allowing users to directly manipulate images, data, and text without leaving ViNE. Currently, we use a persistent-server model for leaf process scheduling but to support systems with low resources, we will provide on-demand processing. For the scientist with very large amounts of data, we will provide more sophisticated ways to organize it, and we need to be able to link directly to existing databases. Adding support for automated maintenance of the tools' configurations described in the notebook is another area of future development. Finally, we will enhance the experiment component, adding facilities for experiment monitoring, steering, and analysis.

References

1. K. Arnold and J. Gosling, *The Java Programming Language*, Addison-Welsey, 1996.
2. Brain Electrophysiology Laboratory.
   `http://hebb.uoregon.edu/brainlab/belHome.html`.
3. J. Cuny et. al. Building Domain-Specific Environments for Computational Science: A Case Study in Seismic Tomography, *International Journal of Supercomputing Applications and High Performance Computing*, Vol. 11, No. 3, pp. 179-196, 1997.
4. D. Edelson, R. Pea, and L. Gomez, The Collaboratory Notebook, *Communications of the ACM*, Vol. 39, No. 4, April 1996, pp. 32–33.
5. A. Geist and N. Nachtigal, Oak Ridge National Laboratory Electronic Notebook Project. `http://www.epm.ornl.gov/ geist/java/applets/enote/`.
6. J. Hong et al., Personal Electronic Notebook with Sharing, *Proceedings of the Fourth Workshop on Enabling Technology: Infrastructure for Collaborative Enterprises*, April 1995.
7. The Math Works, *MATLAB: High Performance Numeric Computation and Visualization Software Reference Guide* Natick, MA. 1992.
8. J. Myers et al., Electronic Laboratory Notebooks for Collaborative Research, *Proceedings of the IEEE Fifth Workshop on Enabling Technology: Infrastructure for Collaborative Enterprises*, June 1996.
9. B. Rex and D. St. Pierre, PNNL NMR Electronic Logbook, *Proceedings of the IEEE Fifth Workshop on Enabling Technology: Infrastructure for Collaborative Enterprises*, June 1996.
10. J. Skidmore, et al., A Prototype Notebook-Based Environment for Computational Tools, *Proceedings of SC98*, November 1998.

This article was processed using the LaTeX macro package with LLNCS style