Notebook interfaces for networked scientific computing: design and WWW implementation

SANJIVA WEERAWARANA 1 , ANUPAM JOSHI 2 , ELIAS N. HOUSTIS 1 , JOHN R. RICE 1 AND ANN C. CATLIN 1

SUMMARY

Advances in wired and wireless networking technologies are making networked computing the most common form of high performance computing. Similarly, software like Mosaic and Netscape have not only unified the networked computing landscape, but they have made it available to the masses in a simple, machine independent way. These developments are changing the way we do computational science, learn, research, collaborate, access information and resources, and maintain local and global relations. We envision a scenario where large scale computational science and engineering applications like virtual classrooms and laboratories are ubiquitous, and information resources are accessible on-demand from anywhere. In this paper we present the design of a user interface that will be appropriate to this scenario. We argue that interfaces modeled on the pen and paper paradigm are suited in this context. Specifically, we present the software architecture of a notebook interface. We lay down the requirements for such an interface and present its implementation using the World Wide Web. A realization of the notebook model is presented for a problem solving environment (PDELab) to support the numerical simulation of PDE based applications on a network of heterogeneous high performance machines. ©1997 by John Wiley & Sons, Ltd.

1. INTRODUCTION

Two parallel developments in computer science are forcing us to rethink the user interface environments of the past, and perhaps move away from a windowing environment, which mimics a physical desktop. One is the development of high performance computing (HPC), which is bringing ever more complex problems from the domain of physical sciences into the sphere of simulation. The most important development in this area, in our opinion, is not the creation of dedicated and fast new supercomputers. Instead, it is the blossoming of networked computing, where software components running on individually small and (possibly) heterogeneous workstations co-operate to solve a problem in a co-operative manner over (possibly) large geographical distances. Sun Microsystem's catchy slogan, 'The Network is the Computer'®, hits the proverbial nail on the head! The other is the development of ubiquitous computing, which is leading to the development of small, handheld computing devices that are mobile, yet constantly connected over wireless networks. While the former increases the complexity and sophistication of the information that has to be made available to the user, the latter restricts the screen space that is available, and also takes away more traditional interfaces like keyboard and mouse and replaces them with a pen-like device. Both these developments are influencing the development of new interface mechanisms for scientific computing.

¹Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398, USA

²Department of Computer Engineering and Computer Science, University of Missouri, Columbia, MO 65211, USA

In the scientific laboratory environment, computational simulation of experimental processes is now an essential component of scientific experimentation. One goal in the area of Computational Science & Engineering (CS&E) is to develop a unified environment where the experimental and computation models which are possibly running on disparate, geographically separated machines, can interact with and complement each other in the problem-solving process[1]. Such an environment would support the construction of mathematical models for the experimental process, the definition of the geometry and associated conditions, the specification of the solution strategy, and finally the generation and validation of results. It would also speak to physical experimentation – allowing the specification of experimental input, process, data acquisition and data analysis in such a way that the information could be used to control the physical equipment in the laboratory. Another important characteristic of this environment would be its ability to access information. Information of all kinds, including that which was traditionally published in archival journals or conference proceedings, is increasingly available on-line. Besides being concentrated in traditional repositories such as libraries, such information also increasingly resides in workstations and computers belonging to individual researchers or research groups, and linked together to form a docuverse[2]. The World Wide Web (WWW, Web) is an example of such a scenario. The fact that some of the citations that you will see in this paper are to Web documents is eloquent testimony to the fact. The environment for scientific computing needs to allow its user to access this information in a seamless fashion. Such an environment has been described as a problem solving environment (PSE)[3].

In this paper, we describe a notebook interface that we have developed for PSEs. It uses NCSA Mosaic as the front end. We introduce in the sections that follow the requirements that such an interface ought to have, and briefly comment on some related work in that context. We then detail the architecture of our notebook and describe the specifics of implementing it along with the supporting system software that we developed. For the purposes of demonstration, we have chosen to integrate some tools developed by other members of our group[4] for the solution of partial differential equations (PDEs). We conclude by discussing this implementation and the work we are currently doing to further improve it.

2. THE WORLD WIDE WEB

Before we go further, however, a brief word on the WWW[5] and Mosaic may be in order here for the uninitiated. The Web is officially described as a 'wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.' It provides users with consistent means to access a variety of media in simplified fashion. Documents for the web are written in HyperText Markup Language (HTML), which is derived from SGML. Documents are thus linked to each other. Every document is identified by a uniform resource locator (URL). A URL refers not only to documents, but can also refer to nearly any service on the Internet. Each networked site supporting the Web runs a server which can understand HyperText Transfer Protocol (HTTP). This server is called an HTTP Daemon[6]. It responds to requests from clients and supplies to them HTML files in its system. Browsers are clients that are used to access the information in the Web. These communicate with servers using HTTP to get the documents, and have the ability to take a document structured in HTML and format it for the user. There are a variety of clients with differing capabilities available, such as Lynx, Netscape, emacs-w3, Arena and Mosaic.

Mosaic is a public domain browser[7,8] which is quite versatile and runs on a number of different platforms (Unix boxes, Macs and Windows). It was developed by the National Centre for Supercomputer Applications (NCSA) and first released in early 1993. When work on this project was first done in the Fall of 1994, it was the most commonly used browser, although it has since been supplanted by Netscape. However, Mosaic is simply a demonstration platform for our work, and does not effect the generality of our results.

3. PROBLEM SOLVING ENVIRONMENTS

A problem solving environment (PSE) is a computer system that provides all the computational facilities needed to solve a target class of problems. These facilities include advanced solution methods, automatic selection of appropriate methods, use of the application's language, selection of appropriate hardware and powerful graphics, symbolic and geometry based code generation for parallel machines, and programming-in-the-large. In this section we describe three different PSE scenarios which serve as the rationale for the requirements we later define for notebook-type interfaces.

3.1. Virtual laboratories

The laboratory presents an ideal environment in which to accept and meet the challenges of computational science and engineering. It is a catalyst for significant research in computational science and engineering, both within the Computer Science Department, as well as in other departments. It accelerates the maturation of a number of collaborative, interdisciplinary research projects that are beginning to create a critical mass of applications-oriented research and expertise in computational science. In the SoftLab[9] project, we are developing hybrid PSEs that are used to create virtual laboratories.

The virtual laboratory is a hybrid PSE that encompasses all research activities occurring in the scientific laboratory environment, both 'wet' and 'dry'. It is a software layer above the experimental and computational processes which exist in the physical laboratory side by side. The virtual laboratory attempts to bring these two models of scientific research together in a way that allows them to interact with each other, so that feedback from one can enhance and improve the methodologies applied in the other. This software layer should support interaction with remote instruments for control of the experimental process. It must also support simulation of the experimental process, which means that it must be able to access and execute the PDE code which models the process. It must act as a training simulator for the scientific methodology, and it should provide scientific visualization of output data resulting from either research model. Since the software layer must handle both models in a unified way, it is clear that they must be tied together from a theoretical viewpoint and from an operational viewpoint. The input and output associated with both models must appear in a uniform way to the scientist using the virtual lab.

We have, in collaboration with colleagues in other departments, developed some virtual laboratories. We now describe one of them – BioSoftLab. Bioseparation is the process of separating chemical components by passing solution mixtures through an adsorbent column. The column is packed with sorbents selected for their adsorption properties with respect to the incoming solutions. Solutions pass through the column and elude from the bottom. Since each solution component adsorbs to the surface of the sorbents in a unique way, components elude at different times. Reactions within the column may also produce new components.

The primary goal of bioseparation is to control what happens in the bioseparation column, so that one can predict when and at what concentration certain components will elude from the column. This process is used for final purification of proteins, chemicals and biochemicals used in the manufacture of pharmaceutical and food products, for water treatment, and for many other biochemical processes[10]. Because bioseparation experiments are both expensive and complex, computer simulation of the process is an attractive alternative to performing experiments. VERSE is a custom implementation of a numerical simulator which models the process, and this code is used by bioseparation scientists to simulate experiments[11]. We have linked this to the SoftLab environment, and a user can now operate both the physical experiment and the numerical simulation seamlessly. The user interacts with the BioSoftLab through an interface which mimics the physical setup of the equipments in the lab.

3.2. Electronic classrooms and mobile/ubiquitous PSEs

The National Information Infrastructure (NII) evolving in the *fin de siecle* has caught the public's imagination, and is popularly referred to as the information superhighway. The superhighway part of the NII will involve very high speed (Gbps and above) connections between major nodal regions. There will also be many state and regional highways. Yet these highways cannot truly provide ubiquitous access to the NII, the kind of access that country roads provide. The 'country road' component of NII will consist of wireless networks used by *walkstations* – hand-held mobile hosts – which will realize the dream of truly ubiquitous access. The NII will impact many aspects of social and academic life; it will engender a continuous interaction between people and interconnected computers using wireless devices.

This new paradigm, variously referred to as mobile, nomadic or ubiquitous computing, raises a host of research issues in a variety of areas of computer science[12]. The nature of the hardware involved in the mobile part of this scenario forces certain restrictions, like limited amounts of communication bandwidth, memory and power. Another critical issue is the nature of the interfaces; the traditional WIMP (window, icon, mouse, pointer) type are unlikely to be successful. These issue must be seriously addressed in the context of mobile computing, since the physical limitations of the proposed devices determine the 'natural' upper bounds on the facilities. We are currently developing *SciencePad*, a PSE which will run on mobile platforms and enable the users to tap into the power of HPCC systems over wireless links.

One can identify three major tools in a scientist's work. Firstly, there is the traditional pen and paper. This is where most of the creative portion of a scientist's work gets done. One writes down ideas, does preliminary designs, plays around with symbolic equations, etc. Then there is the (scientific) calculator. This is used for (relatively simple) numerical and symbolic computation. For more complicated numerical or symbolic calculations, the scientist's uses another tool, the computer. While pen and paper have become ubiquitous tools over the centuries, the other tools mentioned above are still foci of attention. They distract the scientist from the thinking process and force him to divert attention to them in order to put them to proper use. It has been argued[13,14], and correctly in our opinion, that the most efficient tools are those that are invisible, in other words tools that we use without thinking. The objective of *SciencePad* is to serve as such an invisible tool. In order to do this, *SciencePad* provides an interface which enables the scientist to call on

the power of the sophisticated tools like scientific calculators and computers directly and seamlessly. To give an example, consider a scientist working on a design problem which requires an integral to be evaluated. Rather than forcing the use of a calculator or computer, the interface of *SciencePad* recognizes when some symbolic/numeric expression is to be evaluated and it automatically accesses the needed resource, either locally or by contacting some remote computer. This pen and paper kind of model of human–computer interaction involves the design of completely new interface mechanisms[12,14]. Such interfaces will of necessity be multimodal, as well as be endowed with innate intelligence.

Consider a scenario where SciencePad will be used. A researcher usually does some experiments in the lab, then evaluates the presumed theoretical model, or builds a new one, and schedules new experiments to refine the model. SciencePad attempts to aid the scientist in the thinking process, which is usually done using a notepad. A scientist could run an experiment for which s/he has a model involving partial differential equations (PDEs) by controlling the equipment from SciencePad. The experiment can be monitored, and SciencePad can display the results of the experiment in some appropriate form. Then the scientist can make some remarks about the model, change a parameter or just compare the run of the experiment with the results expected from the model. SciencePad can make sense of user remarks (in some constrained and domain specific language) on the basis of previous experience and suggest an action (e.g. solving PDE with the new parameters). Even if a task requires very detailed algorithmic description when it is done the first time, the chances are that, when it is done again, it will require at most minor changes in the parameters. SciencePad provides a framework where such previously defined tasks can be invoked from the scientist's notes. We believe that integrating the problem solving environment interface (in this case, the interface to a PDE solver) with the data from a knowledge base (for example, results from similar experiments or theoretical models for the studied event) into a notebook-like form will enhance the availability, the acceptability and the usability of high performance computing facilities. The intelligent static host solving the PDE can take into account the context sent by the mobile host to display only the relevant parts of the computed results. Ubiquity allows the computations to take place at any place, including in the front-end of the experimental machine, at a discussion table with other colleagues, or at a conference far from the lab. Several important issues involved in designing ubiquitous PSEs are addressed in [15].

3.3. The PDELab PSE

PDELab[4,16] is a problem solving and development environment for PDE based applications. It consists of an intelligent graphical user interface environment for problem specification, problem solution and solution analysis, a high level, symbolic PDE specification language, and a large collection of PDE solver components which can be wired together at a high level to form a custom PDE solver for a given problem. As a development environment, it allows developers to introduce new PDE solving modules, to use the symbolic language to build higher level solver templates, and to study and analyze the behavior of various solvers.

4. USER INTERFACES FOR SCIENTIFIC PSEs

In this section we review existing interface technologies and identify the design specifications of future user interfaces, particularly for scientific PSEs.

4.1. Existing technologies

Very little work has been done thus far on systems that combine information access with computation using a pen and paper based notebook model. Most interfaces are designed around one or the other application. However, work has been done in dealing with the various theoretical issues regarding the development of interactive systems. In [17], Abowd, Coutaz and Nigay describe a structured classification of the properties that should guide the design of interactive systems. Similar issues, arising from a user's view of an interactive system, are presented in the IFIP WG 2.7 report edited by Bass, Cockton and Unger[18]. Apart from such theoretical work, there have also been some implemented systems that provide, in part, functionality similar to what we need. We briefly introduce some of these next.

In [19], Lin presents a simple multimedia editor. He uses the notion of embedded virtual screens, providing one for each medium, and combining them in a pseudo root window. Actually, each different medium is being displayed by a separate client. Clearly, this approach is cumbersome. It expects the user to know what client is needed for which media modality. Also, the user has to directly manipulate the views on screen and place them onto the location where they are to be embedded. This tool is simply an editor, and can neither access information nor be used as a place to launch computations.

Goldberg and Goodisman[20] describe prototypes that explore some issues related to pen based interfaces for text manipulation. They deal with the issue of designing an interface around handwriting recognition software that makes errors, and exploiting the finer control provided by a stylus (compared to a mouse) to do gesture based editing. While interesting from the point of view of the use of such interfaces, their work is confined to text documents alone, and is primarily an editor. It can neither provide information browsing capabilities nor computation facilities.

In an interesting work, Johnson *et al.*[21] present the idea of integrating paper into the input loop for computers. For form based applications, users are asked to use paper to input the data. The paper has special machine readable marks. Scanners are used to convert the paper input into computer readable format, and a special software (XAX server) interprets the images to understand the structure and contents of the document. While innovative, this idea is not appropriate for user interaction with complex systems. In fact, it was specifically designed for form based data entry purposes.

Cohen[22] presents a system which integrates natural language understanding capabilities into a direct manipulation system. It seeks to combine the modes of natural language and direct manipulation into a sequential, synergetic, multimodal interface. The argument Cohen makes is that these two modes have complementary strengths, and their combination can overcome weaknesses that either in isolation has. The paper makes a strong case for some form of (restricted) natural language component to an interactive system.

The information grid[23] is a framework for building information access applications. It views the information retrieval task as consisting of actions by the user, the dialogue machine, the data store and the task machine. In effect, it separates user tasks into an information access stage and a processing stage. In this sense, it comes close to our conception of the notebook. However, its functionality is limited to accepting some search parameters from the user, and then retrieving and browsing through appropriate documents. It does not support the notion of computation independent of operations on information.

A system which augments the web browser capabilities and allows it to display 3D

volume data sets is presented by Ang *et al.*[24]. They extend the Document Type Definition of HTML and use VIS, a distributed volume visualization tool to render the 3D image.

4.2. Requirements and future trends

In order to develop systems that are truly easy to use, PSEs need to provide the user with a high level abstraction of the complexity of the underlying computational facilities. The user cannot, and should not, be expected to be well versed in selecting appropriate numerical, symbolic and parallel systems, along with their associated parameters, that are needed to solve a problem. Since PSEs present an abstraction of the underlying complex system to the user, the choice of the user interface is singularly important. We feel that traditional GUIs are inadequate in this context. Moreover, since PSEs provide a programming-in-the-large environment, the interfaces will need to move away from a direct manipulation approach and towards an indirect management one. With these factors in mind, we posit that electronic notebook like interfaces which use a pen and paper model are the ones most suited for PSEs. Such notebooks move away from the WIMP based interface schemes, and naturally fit into the constraints imposed by subnotebook and personal digital assistant (PDA) type devices. Such devices are likely to become major access points to HPC scientific computing systems in the near future.

The electronic notebook concept is an attempt to emulate the physical notebook that we use ubiquitously. The pen-and-paper physical notebook allows us to write down related or unrelated information in any form we like and to use auxiliary tools (for example, calculators) to augment its usability. The electronic version of this provides an unrestricted editing environment where users can record their activities including problem and solution specifications, computed solutions, results of various analyses, commentary text, graphics and (handwritten) annotations. The notebook interface ideally is multimodal and synergetic; it integrates text, handwriting, graphics, audio and video in its input and output modes. It functions not only as a central recording mechanism, it also acts as the access mechanism for all the tools that support the user's problem solving activities. The electronic version of the notebook is therefore a 'super notebook' when compared to the physical one. It can be viewed as a unifying interface for a collection of software (or other) tools that are needed to define, solve and analyze some problem. Recently, there has been a proliferation of so-called notebook interfaces for various commercial software tools (e.g. Mathematica, Maple, Macsyma, Matlab and HiQ). In most of these systems, the 'notebook' interface is basically a graphical version of the command-driven interface present in the earlier generations of these systems, with additional capabilities for in-line graphics. Also, it is restricted to the particular tool in question. In other words, the notebook for Matlab will not let one access Macsyma, and vice versa.

Given this background, we argue that any interface to PSEs for scientific computing should have the following requirements:

Integrated view of information access and computation. The work of a scientist
requires access to both information sources and computing power. The notebook
interface for PSEs should be able to provide both these functionalities seamlessly.
We are not aware of any current system that takes this approach. As mentioned, commercially available notebooks for various tools are no more than glorified graphical
interfaces. They do not allow the user access to information sources from within.

On the other hand, the many information browsers serve to retrieve information passively; they do not support computations being invoked from within. Browsers for the World Wide Web (WWW) serve to illustrate this point.

- 2. Multimedia/multimodal. With the plethora of information and results being produced by the underlying computational systems, the PSE needs to present the information to users along different dimensions. It also needs to accept input from the user in different formats and interpret it. As such, the notebook should be multimodal in its interaction with the user. Ideally, such a system would be synergetic (i.e. able to combine input from disparate modalities and interpret it) as well, but that is not a very strict requirement for our purpose here. Multimodality differs from multimedia in as much as it is concerned with the meaning of the information flowing through different communication channels, while multimedia is concerned merely with the form[17]. Incorporating true multimodality in the system requires advances in related areas such as natural language processing and speech processing. So as a weaker constraint, we would like the system to have multimedia capabilities.
- 3. Ability to access different tools. A physical notebook does not for the most part restrict what computation is being performed; most anything that the underlying computational system (the brain) can do can be done. We would like a similar functionality from our electronic notebook. This would mean that along the dimension of learnability of interfaces we would like the notebook to be predictable, synthesizable, familiar and generalizable as defined in [17,18]. These characteristics will ensure that users know what operations they will be able to perform on the system, and what changes these will lead to. It will also mean that even though different operations may involve invoking disparate tools, the user's access mechanism will remain the same. This is similar to, but more extensive in scope than, the current efforts by various software vendors (e.g. Microsoft with OLE) to share toolbars amongst different tools.
- 4. Ability to share information between disparate systems. Clearly, if we are going to use different tools in the same system to do operations on the same data, it is necessary that they each be able to make sense of the data. Most existing systems that are linked into a PSE are likely to have native and probably incompatible formats. Our system therefore needs to have some mechanism to exchange this information. Also, it needs to be able to represent these data to users in other words, make the system browsable. This is important because often the data can be voluminous and/or unwieldy from a display point of view.
- 5. *Input/output reuse*. Complex scientific computing systems that will use PSEs will likely have many interacting subparts. Also, the user may wish to have different components work on the same data in different orders. Therefore the interface system should have the capability to reuse user input, as well as the output produced by one tool as the input to another.
- 6. Customizability. While we want our system to retain common look and feel across applications, there is no such need for commonality across users. In fact, given that we want the system to be multimodal, it is evident that some users will prefer one mode over the other. We therefore want the interface to be customizable. This could come from either self-adaptive behavior or user-initiated adaptability.

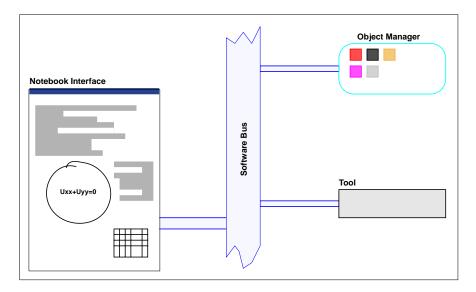


Figure 1. Software architecture of the notebook

5. SOFTWARE ARCHITECTURE OF NOTEBOOK INTERFACES

The requirements described above can be realized in terms of a notebook interface model consisting of the following four major components: the notebook interface itself, the object manager for (persistently) storing (shared) objects, the tools for manipulating the objects in various ways and a communication framework to support the communication needs of these components. In order to realize the generality goals, the software architecture is designed to allow the implementation to be completely independent of the particular types of objects used, the particular tools used and the particular notebook interface used. Object type independence makes this architecture a kernel for building a specific notebook interface for a specific application with specific objects. In this section we describe this software architecture and its implementation using Mosaic.

The basic logical architecture of the notebook model is illustrated in Figure 1. The notebook interface provides all the editing, information access and service access functionality discussed earlier. The object manager provides persistent storage services and allows the notebook to present users with representations of (some of) the objects in the object manager. The tools are application-specific tools that manipulate the objects in various ways. Finally, the software bus is the underlying communication framework that supports all the communication needs of this model.

All interaction between these components occurs in terms of messages between them. The messages generated by the notebook interface include messages to perform certain operations and messages to the object manager to request object representations. Messages sent by tools include messages to the object manager to request objects and to define objects and their representations, and to the notebook to inform it of new objects in the object manager.

In the rest of this section we describe each of the components in more detail and the services provided by or required of them for the realization of the notebook model. We will

use examples and terminology from the PDELab PSE (see Section 7) to illustrate various issues and concepts.

5.1. The PSEBus software bus

The software bus concept is an attempt to emulate the hardware bus mechanism that provides a standard hardware interface to attach additional capabilities to a machine. In the hardware bus, new units describe their capabilities to the bus controller, which then passes the information along to other units in the bus. In our software bus, PSEBus¹, software components register their exported services with the software bus and rely on the software bus to invoke these services when requested by interested clients. The software bus is responsible for the application of any representation translators as required for the valid invocation of the service. Thus, the software bus provides a mechanism where two tools can interoperate with each other without having explicit knowledge about each other and also provides the infrastructure for managing a set of distributed tools. PSEBus supports protocols for at least three different client interactions: the software bus' own client interaction protocol (for clients built with the software bus client library), raw bytestreams (for arbitrary communication) and a line-oriented protocol (for interacting with command-oriented clients). The services provided by PSEBus to clients can be categorized into three groups: location services, process management services and messaging services. For client and object location purposes, a global naming scheme based on uniform resource locators (URLs)[25] is used. The software bus provides various directory services with URLs being the naming standard. The process management facilities provided by PSEBus include facilities to invoke and control both local and remote processes and to set up prewired configurations of clients. PSEBus' messaging services range from low-level byte stream messages to communicating arbitrary data structures via self-describing² and network-transparent representations to remote procedure calls. Using these communication facilities, application specific services (for example, a database service library) can be built for specific needs.

5.2. Bus architecture

The PSEBus architecture reflects the general problem solving environment architectural model that we advocate. In this model, a PSE consists of a collection of tools (clients) that collaborate (under user control) to solve the problem at hand. Clients are built using the PSEBus client library and, at run-time, connect to a *manager* process. A manager process exists for each user, application and machine (an *access domain*) and serves as the clearing house of interclient messages and client requests. While interclient messages travel via the manager by default, it is possible to establish direct, point-to-point links when necessary. (Such direct links may be used when one wishes to communicate (large) data without incurring the cost of travelling through one or more manager processes.) The manager processes themselves are connected to each other via multiple I/O channels. Interprocess

¹The PSEBus is similar to the PDEBus component of PDELab [4,16].

²Self-describing representations allow receivers to interpret raw byte sequences without having prior knowledge of the types of data being communicated. Using such representations makes it possible to build generic services (for example, a tracing facility for monitoring ongoing communication) without being restricted to predefined data types.

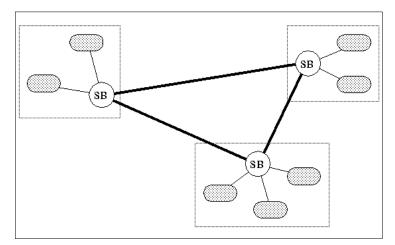


Figure 2. The architecture of PSEBus. SB represents a manager process, an oval shape represents a client while the dotted encapsulating boxes represent access domains

communication occurs via TCP/IP sockets, pipes, shared memory or pseudo-terminals, depending on what the two components can support. In order to avoid being a bottleneck, the manager process generally uses shared memory to communicate with its clients and is multithreaded. Interclient connections are virtual circuit connections implemented over the client—manager, manager—manager and manager—client connections. At any time, a client can have any number of virtual circuit connections to any number of clients (including itself). Figure 2 shows a schematic diagram of this architecture.

The PSEBus implementation respects all the standard access controls supported by the underlying operating environment and guarantees security. This is effected by following the usual mechanisms for getting access to a machine (to run a manager process) and by using a key-based security mechanism for authenticating and validating clients once a manager process is active.

Communication of arbitrary data types is supported in two ways. First, a self-describing data format can be used to inform the underlying communication medium and the receiver of the types of data being communicated. Second, PSEBus allows clients to register their own convertors to/from the data structure and some transport representation. Using this latter mechanism, one can transmit and receive data in the eXternal Data Representation (XDR[26]), for example. A set of utility functions for supporting XDR data communication is included in the current implementation of PSEBus.

The software architecture of PSEBus is a layered architecture with the lowest level providing a packet-based messaging system implemented over a reliable byte-stream protocol such as TCP/IP. The next layer provides support for messaging and arbitrary data type communication. The highest layer provides remote procedure calls and event-driven messaging.

5.3. The object manager

The object manager is the shared workspace that supports information sharing between various tools. The object manager's basic functionality is to provide persistent storage

capability to the notebook and other tools. Objects are named using uniform resource locators and have associated with them a representation property to maintain various representations for them. Tools are also allowed to register interest in certain types of events and the object manager generates messages to the tools to inform them of such events. Since multiple references are allowed to objects, a reference counting scheme is used to identify active and garbage objects.

The object manager also includes a graphical user interface based on a file system browser model to allow users to perform various actions on the objects directly. These actions include renaming, deleting and performing type-specific actions.

Persistency is achieved by storing objects in the file system and accessing them during each session. Objects may be shared between (collaborating) sessions by joining the databases of two or more managers.

5.4. The notebook interface

The notebook interface is expected to provide much of the user-level functionality that we outlined earlier. These include the pen and paper interaction model, word processing type capabilities, and intelligence as well as information browsing capabilities (based on the web, for example). In addition, it must allow users to include representations of objects produced by various tools. Each object that is represented in the notebook is typed. At start-up, the notebook is configured with the set of types of objects, the possible display representations of each of these types as well as the list of valid type-specific actions. The display representations are named using MIME syntax[27].

Using this model a session of the notebook interface contains, in addition to arbitrary text and images, a collection of references to objects maintained in the object manager. These references are used to display to the user some representation of the object (based on the list of representations available for each object). The notebook interface is itself application independent; an application specific notebook is built by configuring the generic notebook environment with the types of objects that are used in the application. For each type, the configuration information also lists the valid representations for that type and the list of actions that one can perform on objects of that type.

5.5. Messages for implementing the notebook model

The notebook is implemented using a small number of well-defined messages between the various components involved. Tools added to this environment must service the following three messages:

- edit [object_URL]
 - When this message is received, the tool starts in editing mode. If an object URL is specified, then that object is loaded from the object manager for editing. Any saved changes affect the object stored in the object manager.
- view [object_URL]
 When this message is received, the tool starts in viewing mode. That is, the user is allowed to manipulate the object for viewing purposes only. Any changes may not be saved in the object manager.
- do args

This message is used to request all other services from tools. For example, sending the message 'do mesh domain_object_URL' to a mesh editing tool results in the mesh tool loading the given domain object and generating a mesh on that domain. The arguments and semantics of the 'do' messages are dependent on the first argument of 'do'.

The object manager services the following messages:

• put object_URL object_data

This message is used to insert an object into the object manager. The object data may be opaque to the object manager and is simply associated with the given URL. If the object already exists, then this request must originate from the same client currently holding the key to this object. Updating an existing object invalidates all read-only copies given out earlier.

• get object_URL mode

This message is used to request an object. The mode indicates whether it is 'read-only' (i.e. for viewing purposes) or for 'read-write'. Only one client may have any given object checked out for read-write access. The result is sent to the requestor as a 'get-reply' message.

• delete object_URL

This message is used to delete a reference to an object. If the reference count of the object goes to zero as a result of a delete message, that object is then marked for possible destruction.

• putrep object_URL rep rep_data

This message is used to define some representation of an object to the object manager. The representations are associated with the object URL for later use. If a certain representation already exists for the given URL, then the new one overrides the old one.

getrep object_URL rep1 rep2 . . .

This message is used to request some representation for an object in any one of the representations listed. The first available representation is returned.

The notebook services the following message:

• put object_URL display_type

This message is used to tell the notebook that a new object is available in the object manager. The default action is then to retrieve an appropriate representation for the object and to display that in the notebook. The display type is used to locate the set of available representations for the given object.

All of these messages are implemented in terms of encapsulated library functions. That is, applications generate these requests by invoking library functions that make the appropriate connections via the software bus to forward the request.to the appropriate clients. The messages are serviced using the software bus' message delivery facilities.

Using these messages, the notebook model is implemented as follows:

1. When a tool is invoked, it is sent either the 'edit', 'view' or 'do' messages with the appropriate arguments. If an object URL is given as an argument, then that object is retrieved from the object manager by sending a 'get' message to it.

2. When an object is saved by a tool, it sends a 'put' message to the object manager to define the object and then sends one or more 'putrep' messages to define various representations for it. Then, it sends the 'put' message to the notebook to inform it about the new object available in the object manager.

3. When the notebook receives a 'put' message, it determines the possible representations for the given display type and then sends a 'getrep' message to the notebook to request one of these representations.

6. IMPLEMENTING THE NOTEBOOK MODEL USING NCSA MOSAIC

As has been mentioned earlier, the WWW is probably the most popular form of storing and accessing information. To realize the notebook model using NCSA Mosaic, the notebook interface must implement the 'put' message described earlier and also have a mechanism to generate the 'edit', 'view', 'delete' and 'do' messages to tools and the 'getrep' message to the object manager. In this section, we describe two possible approaches for implementing this approach using NCSA Mosaic – one requiring no changes to it (but requiring an HTTP daemon) and the other requiring some changes. An important guiding principle in our design was to limit the modifications/customizations necessary to realize the notebook model using NCSA Mosaic. In both approaches that we considered, a notebook session is represented by an annotated HTML file so that we can take full advantage of the multimedia and formatting facilities provided by NCSA Mosaic. The annotations are represented as specially formatted HTML comments and are interpreted by a special preprocessor which uses this session to generate a customized HTML file to give to NCSA Mosaic.

Using NCSA Mosaic, however, does introduce some limitations with respect to the ideal notebook system model described earlier. Most importantly, the current NCSA Mosaic system does not support *in situ* editing and hence the word processing features we required are not yet possible. In-place editing is, however, evolving for Web browsers (for example, Arena[28]) and we can capitalize on this when it becomes available as our design deliberately avoids requiring a specific Web browser. The same limitations also arise for the intelligence requirements mentioned earlier; current browsers do not support integrating such intelligence mechanisms. Very recently, Netscape has unveiled an in-line plug-in facility which does allow for such integration, and we hope to exploit it in our future work. More generally, while the implementation described here uses Mosaic, Netscape could be integrated into our system as the browser of choice using fairly straightforward modifications. This is facilitated by Netscape's runtime API. Infact, most browsers provide mechanisms similar to the ones we have exploited for Mosaic.

We will first consider the implementation using a custom HTTP daemon.

6.1. Implementation using an HTTP Daemon

In this approach, no modifications to NCSA Mosaic are needed. Instead, its ability to be controlled via the SIGUSR1 signal is used to produce the required behavior. When NCSA Mosaic receives an SIGUSR1 signal, it checks for the existence of a file named /tmp/Mosaic.<pid> (where <pid> is the process identifier of the NCSA Mosaic process). If the file exists, then the first two lines in the file are read in and evaluated; the first line is assumed to be a command (such as 'goto', 'reload', 'open' or 'close') and the second line is assumed to be the URL for that command (if needed). When a new session starts,

an initial session HTML file is generated and displayed in the NCSA Mosaic window. The architecture of this implementation is shown in Figure 3.

The primary notebook functionality is implemented in this approach by the notebook driver and not the NCSA Mosaic interface itself. Hence, as far as all the tools and the object manager are concerned, the notebook driver *is* the notebook interface. To effect this, when the driver receives a 'put' message, it updates the session HTML file by contacting the object manager to obtain a representation for the object and inserting it in the session file. Then, NCSA Mosaic is instructed to reload the current file (via an SIGUSR1 signal) and the saved object appears in the notebook interface. In addition to the object representation, the driver also inserts hyperlinks that allow the user to perform various actions on the object (for example, edit, view and delete). These actions are generated from a table (read in at start-up) that lists the valid actions and how the actions are to be implemented for each display type. The display type of an object is given to the notebook (driver) when the put message is received.

Each action must be implemented via a message to a tool or the object manager. Since actions are initiated when users select hyperlinks in the generated HTML file being viewed in NCSA Mosaic, they are represented as HTTP URLs directed to a special HTTP daemon that is running for this session. These URLs are of the following form:

http://hostname:port/PSEBus/clientname/message/argcnt/arg1len/arg1/... where hostname and port identify the specific HTTP daemon. This special server is configured to translate URLs of this form to a script that is invoked via the common gateway interface (CGI)[29]. By parsing the extra arguments, the PSEBus gateway script determines the client to be contacted as well as the message (including arguments) to be delivered. Finally, it connects with the software bus and delivers the message, thereby effecting the action that the user initiated.

This approach has the significant advantage that it is implemented with the stock NCSA Mosaic³ browser and any CGI-compliant HTTP daemon. However, the need to run a CGI binary almost always requires one to run a special daemon configured with this path alias. Running an HTTP daemon for each session also requires the session manager to choose a random port for it. Because of the rather complex execution model of this approach, we developed an alternative method that does not require running an HTTP daemon, but does require some (fairly simple) modifications to NCSA Mosaic.

6.2. Implementation with NCSA Mosaic modifications

In this approach, we eliminate the need for an HTTP daemon by modifying NCSA Mosaic to directly interact with the software bus to implement the actions discussed earlier. The resulting architecture is illustrated in Figure 4. The mechanism for remotely controlling NCSA Mosaic and the mechanism for evaluating actions was changed in this approach.

When the notebook driver generates hyperlinks for the actions, rather than generating HTTP URLs directed to a special server, it generates URLs of the following form:

psebus://clientname/message/argcnt/arg1len/arg1/...

where 'psebus' is a special URL type that is recognized by the modified NCSA Mosaic. The implementation of this URL type in NCSA Mosaic parses the URL to obtain the client

³NCSA Mosaic itself is not needed either; this approach can be implemented with any browser that supports some remote control mechanism.

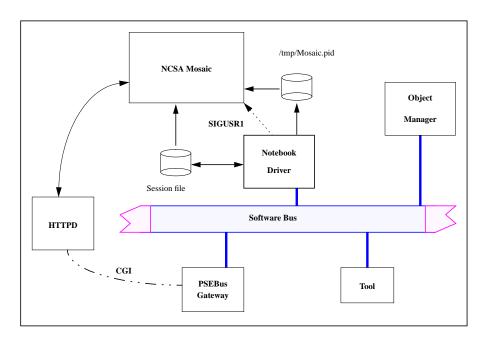


Figure 3. Architecture of the HTTPD-based notebook system

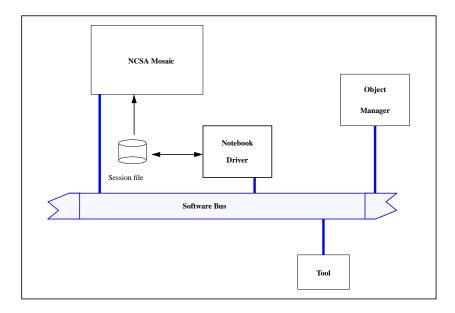


Figure 4. Architecture of the notebook system based on a modified NCSA Mosaic

name and message information and then contacts the client directly via the software bus to deliver the message.

The result is a much-simplified execution model as well as improved performance as the number of links involved in processing actions has been reduced. Furthermore, as discussed in the section below, even the modifications we made to NCSA Mosaic may be unnecessary once the common client interface[30] is implemented by various browsers.

7. THE PDELab PSE

In this section we describe a partial implementation of PDELab using the environment discussed in this document. The NCSA Mosaic based notebook interface for PDELab is built by writing configuration scripts that instruct the interface about the particular application. We use the Tcl language[31] for writing configuration scripts as that is a very effective mechanism for writing complex configurations for applications.

```
# all the tools
set tools {{eqn
                  Equation
                              egntool
                                             equation.xbm}
           {dom
                              domaintool2d domain.xbm}
                 Domain
                 Grid
           {grd
                              gridtool2d
                                             meshgrid.xbm}
           {dec
                 Decompose
                              decomptool2d
                                            decomp.xbm}
           {alg
                  Algorithm
                             algorithmtool algorithm.xbm)
           {cmp
                 Composer
                              solvetool
                                             compose.xbm}
           {exc
                 Execute
                              executetoo1
                                             solve.xbm}
           {ana
                 Analyze
                              outputtool
                                             analyze.xbm)
                 NLPTool
                              scipad
           fgaa}
                                             foo.xbm}}
foreach i $tools {
    # create the button, add the bitmap (if available) and pack it
   button $tl.[lindex $i 0] -text [lindex $i 1] \
           -command [list NbdPSEBusCmd [lindex $i 2] "edit" $sessionid]
   if {[file exists $appdirectory/bitmaps/[lindex $i 3]] == 1} {
        $tl.[lindex $i 0] configure \
               -bitmap @$appdirectory/bitmaps/[lindex $i 3]
   pack $tl.[lindex $i 0] -side left
```

Figure 5. Configuring the set of tools available via the notebook

```
set standardreps {image/gif text/plain}
# define all the types involved in this case and also the actions they
# are supposed to support. "<OBJECT>" will be replaced by the object's
# url at run time.

set actions {
    {Delete notebook delete <OBJECT>}
    {Edit domaintool2d edit <OBJECT>}
    {View domaintool2d view <OBJECT>}
    {"Define Boundary Conditions" domaintool2d do bc <OBJECT>}
    {"Generate Grid" gridtool2d do grid <OBJECT>})
type domain2d $sessionid $standardreps $actions
```

Figure 6. Configuring the 'domain2d' data type

Figure 5 shows a part of the PDELab notebook configuration file where the set of tools is specified. For each object type, possible display representations and possible actions must also be listed. Figure 6 shows how the 'domain2d' type is configured. The object manager is configured by giving it the various transport and I/O functions needed for dealing with the new type of data object. The OM uses these functions to transport objects between various components of the system and also to persistently store them in its database. This configuration is done in C as the functions must be explicitly linked in to create an application specific object manager binary.

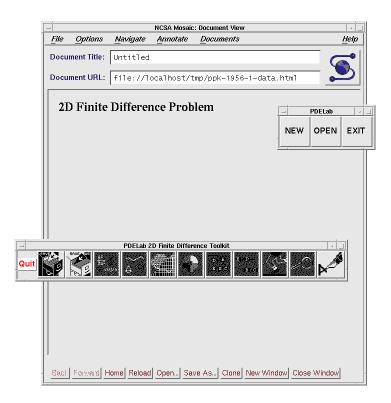
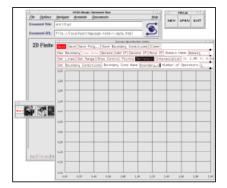


Figure 7. Notebook at start-up

Figure 7 illustrates the notebook system as it starts up where the toolbox shown is dynamically generated using the information from the configuration file. Figure 8 shows the invocation of the domain specification tool and the result of saving the domain from it. The tool was invoked by clicking on the appropriate icon, which resulted in the appropriate message being sent to the tool. After the domain was drawn in the domaintool it was saved to the notebook via the process described earlier: the tool places the object in the object manager and then informs the notebook about it. The notebook (in this case, the notebook driver which drives the NCSA Mosaic interface) responds to this information by contacting the OM and requesting a valid representation for the object and then by



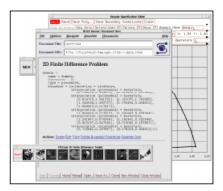


Figure 8. Accessing the domain tool from the notebook

generating an annotated HTML file which is displayed in NCSA Mosaic. The 'reference' links which allow users to easily invoke actions on the objects are generated as URLs of the form 'psebus://', which are in turn evaluated by our modification of NCSA Mosaic. The formats of these action implementations are also read in from the configuration file.

Using the infrastructure described in this paper, we have thus built a prototype version of PDELab with a multimedia notebook interface in a very short time.

8. CONCLUSION

In this paper, we have presented a web based notebook interface that was designed to operate with problem solving environments for scientific computing. This interface satisfies the criteria that are important for PSEs. To wit, it provides a seamless integration of information access and computation. This is achieved by consciously choosing Mosaic as the front end of the system, and modifying it to work with a software bus based architecture to access computational tools. This allows the user to access information through the Web, as well as the other wide area information retrieval protocols that Mosaic subsumes, such as ftp and gopher. The interface, while not truly multimodal, certainly supports multimedia. A small degree of multimodality is provided by adding restricted natural language interfaces for the application at hand. We are working on extending this natural language system, and integrating it with a visual system to recognize shapes. Commercial handwriting recognition software will also be integrated into the system. Using the software bus architecture allows us to access disparate tools from within the same interface and using a consistent point and tap (or click, for a mouse) interface. The self-describing and network transparent representations formats allow us to share data between disparate tools with incompatible native formats. Since all data are represented in some format on the notebook, reuse of input and output is simply a matter of selecting it on screen. Mosaic can also be customized in its look and feel by different users. In addition, each object is allowed to have multiple onscreen representations, and the user can choose the one they prefer to have displayed. This makes our system adaptable. We are working on making it self-adaptive. In other words, we want the system to automatically choose representations based on three parameters: the user's preferences, the resource constraints and the object it seeks to display.

REFERENCES

 E. N. Houstis, A. Joshi, J. R. Rice and S. Weerawarana, MPSE: Multidisciplinary Problem Solving Environments, White Paper presented at the CIC Forum on America in the Information Age.

- CERN, How to put your Data on the Web http://info.cern.ch/hypertext/WWW/Provider/Overview.html.
- 3. E. Gallopoulos, E. N. Houstis and J. R. Rice, 'Computer as thinker/doer: Problem solving environments for computational science', *IEEE Comput. Sci. Eng.*, **1**, 11–23 (1994).
- S. Weerawarana, E. N. Houstis, J. R. Rice, A. C. Catlin, C. L. Crabill, C. C. Chui and S. Markus, 'PDELab: An object-oriented framework for building problem solving environments for PDE based applications', *Proc. Second Annual Object-Oriented Numerics Conference*, Rogue-Wave Software, 1994, pp. 79–92.
- T. Berners-Lee, R. Cailliau, J.-F. Groff and B. Pollermann, 'World Wide Web: The information universe', *Electronic Netw. Res. Appl. and Policy*, 2, (1), 52–58, Spring 1992.
- 6. NCSA, NCSA httpd Overview, http://hoohoo.ncsa.uiuc.edu/docs/Overview.html.
- M. Andreessen and E. Bina, 'NCSA Mosaic, A global hypermedia system', *Internet Research*, 4, (1), 7–17 (1994).
- 8. NCSA, NCSA Mosaic Home Page, http://www.ncsa.uiuc.edu:80/SDG/Software/Mosaic.
- 9. C. M. Hoffmann, E. N. Houstis, J. R. Rice, A. C. Catlin, M. Gaitatzes, S. Weerawarana, N.-H. L. Wang, C. G. Takoudis and D. G. Taylor, 'SoftLab A virtual laboratory for computational science', *Math. Comput. Simul.*, **36**, 479–491 (1994).
- 10. K. E. Van Cott, R. D. Whitley and N.-H. L. Wang, 'Effects of temperature and flow rate on frontal and elution chromatography of aggregating systems', *Separat. Technol.*, **1**, 142–152 (1991).
- 11. J. A. Berninger, R. D. Whitley, X. Zhang and N.-H. L. Wang, 'A versatile model for simulation of reaction and nonequlibrium dynamics in multicomponent fixed-bed adsorption processes', *Comput. Chem. Eng.*, **15**, 749–768 (1991).
- 12. M. Weiser, 'The computer for the twenty-first century', Sci. Am., 268, 94–104 (1994).
- 13. M. Weiser, 'Some computer science issues in ubiquitous computing', *Commun. ACM*, **36**, 75–85 (1993).
- 14. M. Weiser, 'The world is not a desktop', ACM Interactions, 7-8 (1994).
- 15. T. Drashansky, S. Weerawarana, A. Joshi, R. Weerasinghe and E. Houstis, 'Software architecture of ubiquitous scientific computing environments for mobile platforms', *CSD-TR-95-065*, Department of Computer Sciences, Purdue University.
- S. Weerawarana, Problem Solving Environments for Partial Differential Equation Based Applications, PhD. thesis, Department of Computer Sciences, Purdue University, 1994.
- G. Abowd, J. Coutaz and L. Nigay, 'Structuring the Space of Interactive system properties', Proc. EHCI '92.
- L. Bass, G. Cockton and C.Unger, 'IFIP WG 2.7 user interface engineering: A reference model for interactive system construction', *Proc. EHCI* '92.
- 19. J. K. Lin, 'MediaMosaic A multimedia editing environment', Proc. UIST '92, pp. 135-141.
- D. Goldberg and A. Goodisman, 'Stylus user interfaces for manipulating text', *Proc. UIST* '91, pp. 127–135.
- 21. W. Johnson, H. Jellinek, L. Klotz, R. Rao and S. Card, 'Bridging the paper and electronic worlds: The paper user interface', *Proc. INTERCHI* '93, pp. 507–512.
- P. R. Cohen, 'The role of natural language in a multimodal interface', Proc. UIST '92, pp. 143– 149.
- 23. R. Rao, S. Card, H. Jellinek, J. Mackinlay and G. Robertson, 'The information grid: A framework for information retrieval and retrieval-centered applications', *Proc. UIST* '92, pp. 23–32.
- 24. C. S. Ang, D. C. Martine and M. D. Doyle, 'Integrated control of distributed volume visualization through the World-Wide-Web', *Proc. Visualization* '94, 1994, pp. 13-20.
- T. Berners-Lee, Uniform Resource Locators: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network, http://info.cern.ch/hypertext/WWW/Addressing/URL/Overview.html.
- 26. Sun Microsystems Inc., Network Programming: External Data Representation Standard, 1990.

- 27. N. Borenstein and N. Freed, MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, Internet RFC 1521.
- 28. Arena: W3C's HTML3 Browser, http://www.w3.org/hypertext/WWW/Arena.
- 29. NCSA, *The Common Gateway Interface*, http://hoohoo.ncsa.uiuc.edu/cgi/index.html.
- 30. NCSA, NCSA Mosaic Common Client Interface (CCI 1.0), http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/CCI/cci-spec.html.
- 31. J. Ousterhout, 'Tcl: An embeddable command language', *Proc. USENIX Winter Conf.*, 1990, pp. 133–146.