Information Networking for Distributed Semiconductor Technology Development

by

Chantal E. Wright

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology
May 28, 1996

Author_____

Department of Electrical Engineering and Computer Science
May 28, 1996

Certified by_____

Duane S. Boning
cal Engineering
ıesis Supervisor

Accepted by_____

Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Information Networking for Distributed Semiconductor Technology Development

by
Chantal E. Wright

Submitted to the
Department of Electrical Engineering and Computer Science

May 28, 1996

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering and
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This work explores the application of the World Wide Web and the Java programming language to various modes of information sharing for the support of distributed semiconductor research and development. We begin by describing the utility of a repository, local or distributed, of static documentation. Next we discuss the importance of making dynamic processing data available throughout a community of researchers and describe an implementation which provides data from MTL's CAFE system to coresearchers. We mention the absence of a well-used academic data representation standard for process information and the benefit which could be derived from the implementation thereof. Finally, we discuss an implementation of a process flow editor which anticipates such a standard and thus demonstrates the possibility of geographically distinct researchers making immediate use of a shared pool of data for realizing distributed process design, simulation, and fabrication.

Thesis Supervisor: Duane Boning
Title: Assistant Professor of Electrical Engineering

Acknowledgments

I am sure that the production of this thesis would have far more difficult and much less pleasant without the continuous support and help I had from many people.

First, I thank Professor Duane Boning. Duane has consistently been an effective and creative teacher and guide. He is clear and helpful, and always presents his ideas and suggestions in a constructive and positive manner. I am grateful for the opportunity I have had to work in his group.

Many heartfelt thanks to my mom, for her constant encouragement and support throughout my 24 years. And to my dad for many daytime conversations in the past six years. Thanks to my brothers and sister and Daisy for being entertaining. And thanks to all of my friends, especially Heather, whose steady stream of email has made the last year of staring at the computer screen bearable.

I would still be struggling with some of the problems I encountered during this work if it weren't for Aaron Gower and William Moyne; many thanks for their time, knowledge, and insight. Thanks to Taber Smith and Ka Shun Wong for the frisbee games and lunches that kept the office lively and fun.

Finally, thanks to Paul Calame, Damon Apostalon, Richard Crowley, and Don Currier at Intel for their support and ideas during my internship there.

This research has been supported by ARPA under a subcontract with Stanford and the Computational Prototyping for 21st Century Semiconductor Structures.

Table of Contents

List of Figures

Chapter 1

Introduction: The Web and Semiconductor Technology Development

The concept of distributed research and development is not a new one; however, its implementation using the World Wide Web and related network technologies has only just emerged as a real and attractive possibility. The goal of this work is to explore and evaluate this possibility in the context of semiconductor process research and design.

Effective distributed research can promote and speed progress in any field by increasing the pool of manpower and resources. In the particular field of academic semiconductor research, collaborative effort has long since been required; due to the expense associated with fabrication equipment, only a subset of the equipment in a university fabrication facility is state-of-the-art. Often, one university has access to a unique resource and only by collaboration may outside researchers utilize that resource in their work. The Web and associated capabilities may provide the features that will enable the simplified and consistent information and resource sharing necessary to realize effective distributed research.

1.1 World Wide Web

The World Wide Web was introduced in 1990 by CERN, the European Particle Physics Laboratory in Geneva, Switzerland. The Web was developed to allow geographi-

cally separate collaborators to share their ideas and work on a common project. When the supporting software became widely distributed, anyone with an IP address could install and manage a "Web Server" and thus provide a static presentation of information to anyone with Internet access and a "Web Client," commonly called a browser. The Web has gained unprecedented widespread use as a tool for simple data sharing among remote sites. Several features of the Web have made this possible.

The Web's support of hyperlinks is perhaps the most obvious contributor to its rapid growth. Hyperlinks allow arbitrary networks of information to be connected and traversed. An information provider can create local links within his own repository as well as link his information to other repositories. These links can point to anything that can be displayed, allowing a user to browse through multiple types and presentations of data.

A second important aspect of the Web is that clients provide an identical user interface to most Internet protocols. Regardless of the platform or server type, a user only need be familiar with this single interface. Furthermore, since this tool can be used to access and retrieve data in different formats from arbitrary repositories, users do not need to acquire multiple tools or learn multiple access and retrieval methods.

Web clients support presentation of some data formats such as GIF and JPEG within the browser; they also provide the option of automatically spawning locally available "helper applications" to display other formats. The result is that virtually any computer accessible information can easily be made available on the Web, whether in native format or converted to HyperText Markup Language (HTML), the markup syntax associated with authoring Web pages.

In considering applications of the Web technology, it is important to note that the core capabilities of a Web client are limited to the retrieval and presentation of static pages or files of information. The only dynamic capability of the Web (as initially introduced) relies upon the Server. Web Servers, such as the NCSA HTTPd, have a semi-dynamic feature associated with using the Common Gateway Interface (CGI). It is possible for the Server to return to a requesting client the standard output of a script or program run by the Server in the CGI.

The Web client which requests that the output of a program be returned can send parameters along with its request. In this way, the Web Server can provide a dynamic pre-

sentation of information to the user. Information returned by a program may be, for example, information extracted from a database on the fly, or a response to inputs provided by the user via an HTML form.

After the static browsing capabilities of the Web had stabilized and become useful, Sun Microsystems introduced the Java programming language which has revolutionized the power and potential of the Web. Java programs can be downloaded to a client machine and run locally. This gives the Web a mechanism for dynamic presentation; complex programs involving user interaction may be embedded in an information provider's Web page.

1.2 Sun Microsystem's Java

Sun introduced Java as a programming language for the Internet in 1995 [Gos95]. At the time of this writing, Java 1.0 has been released as the first official version. There is currently widespread experimentation with the language among programmers interested in providing dynamic capabilities to Internet users. The language is poised to become an integral part of the World Wide Web.

The syntax and power of Java is similar to C/C++. It is an object oriented language which is compiled into an architecture-neutral bytecode. Compiled Java programs may be referenced in HTML pages as "applets." A Web client that supports Java will retrieve Java bytecode based on the HTML reference, and then interpret and run the applet on the client machine.

Because the bytecode of a single compiled Java program may be interpreted and executed on any client platform without modification, Java may be considered a "universal" Internet programming language. Java applets are written without concern for the system upon which they might ultimately run. It is the system specific Web clients (such as Netscape) that are responsible for interpreting the Java bytecode and running it on the local platform.

With Java applets, Web pages become dynamic and interactive. Information providers can provide static documents as before, but Java introduces the capability to (safely) run programs on the browsing machine. Old programs written, for example, in

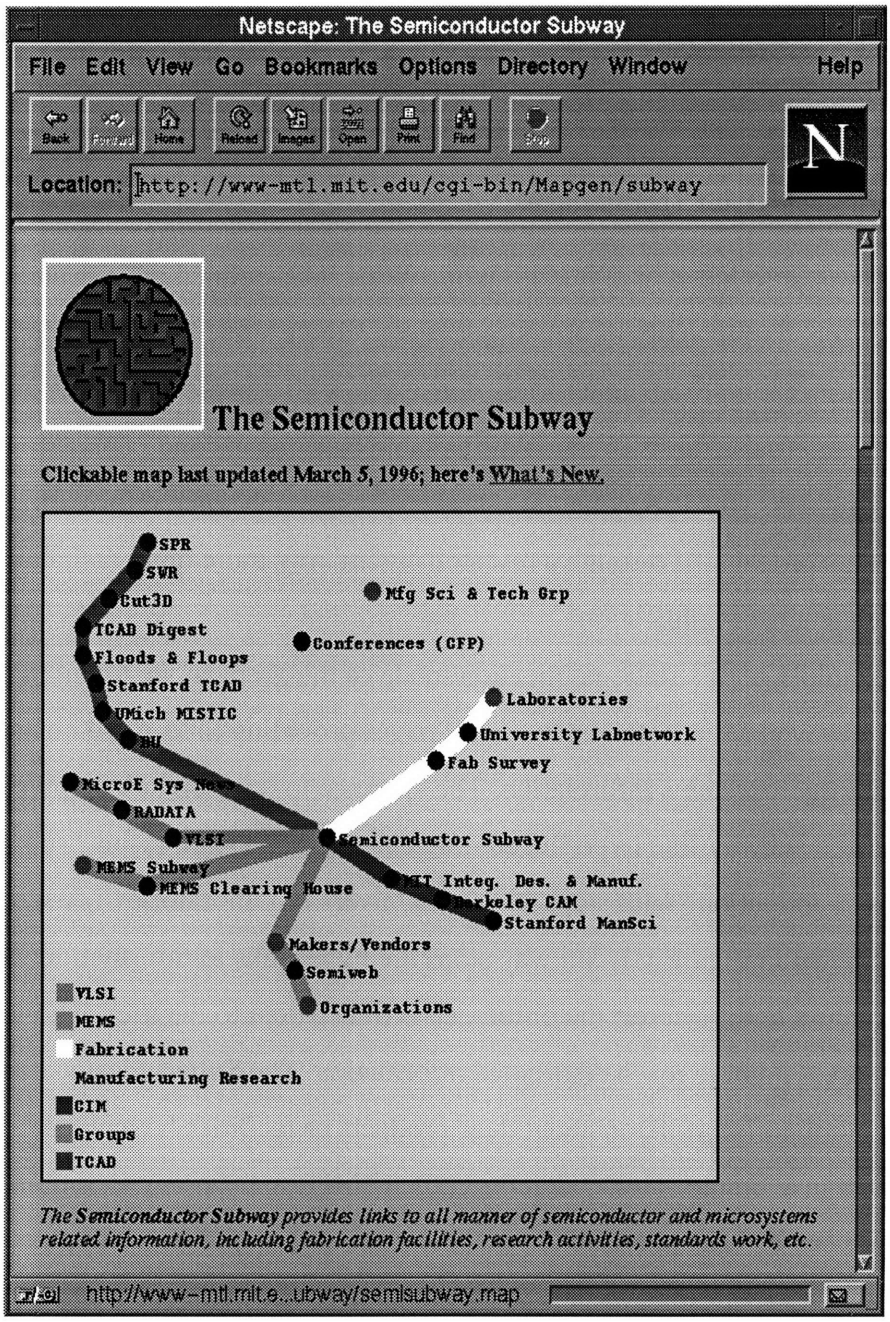
Tcl/Tk and ported to different machines can be written in Java once and immediately available for execution by a community of users on the Web; a user will not need to install any specialized software, she will simply need a Web browser.

1.3 Applications of Web Technologies

As it was introduced in 1990, the World Wide Web defined an interface and standard protocol for sharing static information among remote sites. In any research environment, much information exists in a format consistent with this presentation mode; researchers began to use the Web to make available their results and papers to a broad community of peers. This is beneficial to distributed research since information and updates are thus propagated through a research community more quickly than through the established channels of publication.

An example of this static publication mode of the Web can be found in the Semiconductor Subway at MIT, a static network of pages providing links to all manner of semiconductor and microsystems information. The entrance to this network is a Web page which presents a subway-like hyperlinked imagemap to each visitor; this "subway" has lines such as "Fabrication" with stops such as "University Laboratories." A screen shot of this page is shown in Figure 1.1. Subway stops are hyperlinks to information which may be local or remote to the Server; the difference is transparent to the user, who visits one centralized access point to reach distributed information. The Subway is updated as people request that a link to information about their products or facilities be included. All links are maintained in a simple subway definition file at MIT; a Tcl program running in the Server's CGI uses that file to create the graphical subway map. This example, then, illustrates the use of a local tool (a local program running in the CGI) and a small amount of centralized maintenance to provide users throughout the world easy access to a wealth of distributed information.

Figure 1.1 The Semiconductor Subway



Making available information and results is certainly an important step toward advancing research across a large geographical front, but distributed research demands distinct parties working in concert toward a common end. This type of interaction requires more information sharing than static document availability can provide. The initial Server-based dynamic CGI capabilities of the Web were put to use with this in mind when

researchers began to make various dynamic on-line resources such as databases more widely available. Increasingly ground-breaking examples of sharing unique resources lie in projects at Stanford to provide Web access to semiconductor process simulators and at MIT to interface unique fabrication equipment to the Web [Los96]. Finally, Java introduces the possibility of implementing truly distributed and dynamic applications; programs will be passed across the network and will access arbitrary information resources.

We have explored several specific applications of these technologies in this work. First, we have examined the use of the Web and its basic CGI functionality to automate a process for the organization and presentation of the wealth of "miscellaneous" information associated with processing and development in the field of semiconductor research. In developing this process, we have explored a paradigm of group publishing of semi-dynamic (regularly updated) information. This phase of the work, presented in Chapter 2, was undertaken during a summer intern assignment at Intel's Portland Technology Development, the process development group.

Next, we explore the application of the technology to well-defined semiconductor data resources. Specifically, we employ the CGI to allow coresearchers to view and traverse our semiconductor processing database. This is an example of dynamic information sharing; a distant researcher can view data describing objects such as process flows, lots, and lab users associated with MIT's Microsystems Technology Laboratories (MTL). This information will give that user an up-to-date view of activities in the lab, as opposed to that researcher waiting for the publication of relevant updates. This system, described in Chapter 3, provides database browsing capability, but does not address the issue of actual data sharing wherein geographically distinct researchers access and utilize the same databases.

One obvious requirement for shared resources is a common mode of data communication. In industrial technology transfer, a standard process description mechanism is typically well-established within each company. In academia, the Semiconductor Process Representation (SPR) is an emerging standard. Since the SPR is not yet in widespread use, we are prevented from developing a system whereby researchers could immediately share data. In this research, we have developed a system which attempts an initial implementation of the SPR in the Java programming language and then supports an SPR-based dis-

tributed process flow development and design environment.

At MTL, the Computer-Aided Fabrication Environment (CAFE) includes a Tcl/Tk based graphical process flow editor. This tool allows CAFE users to retrieve, examine, and build process flows from the associated process library. This editor is restricted to local usage first by its system-specific implementation, and second by its fundamental dependence upon CAFE and its Process Flow Representation (PFR). We have implemented a Java applet (presented in Chapter 4) which maintains the basic functionality of this useful tool, but introduces the potential for an increased research community to access process information from multiple process libraries across the network. Since it is written in Java, any user with a (Java-capable) Web browser could access it. Since it implements the standard semiconductor data representation (SPR), it may potentially be used by remote researchers to create a pool of shared process information and thus make a significant contribution to realizing distributed research in semiconductor process design.

The development, implementation, and success of the application of Web technologies to the three problems outlined above will be described in the subsequent chapters of this thesis, and conclusions will be offered in Chapter 5.

Chapter 2

Organization and Presentation of Unstructured and Static Information

Many large companies such as Intel have taken great interest in exploring the potential use of the Web as both a vehicle for an external presentation of the company to the public and a means to facilitate internal distribution of information. I spent several months at Intel investigating the latter possibility: the internal dissemination of data and information to support technology development.

In the semiconductor industry, as in other fields, there is a substantial quantity of unstructured or miscellaneous information which is important and relevant and must be propagated throughout appropriate groups. Examples of this type of information include reports and updates meant to be widely available, such as weekly status reports. Such information is historically distributed by somehow supplying a copy to each intended recipient. There are several clear disadvantages to this method. First, the immediate result is that tens or even hundreds of people will each have a paper or an electronic copy of the information. A paper copy is likely to be lost or discarded and hence unavailable for future reference. Further, repeatedly providing large numbers of people with paper copies is wasteful. An electronic copy may be stored or deleted; if it is stored, the result is that every recipient is using disk space to maintain her own copy. In either case, if revisions are made to some document and redistributed, it is likely that there will be confusion regarding which is the most recent version: there is no longer any assurance that everyone is refer-

encing the same version of a document.

An obvious solution to these problems is to maintain a single active and official version of such a document at some well-known location. Document Control divisions of companies like Intel have long been responsible for doing just this with hard copies of important and highly confidential information. Less restricted information like procedural specifications may be maintained and available on-line through some well-defined and specific interface. This solves the problems described above, but it suggests a new difficulty. If information were made available on-line such that each type or category of information were accessible through its own specialized interface, there would be too many different access methods to remember to find anything.

The World Wide Web may provide the solution to this interface problem. Repositories of arbitrary information may be maintained and made accessible on the Web. This means that the user will need to be familiar with only one interface: the Web browser. To locate particular information, one would need either to know the URL at which the data resides or to be provided with a straightforward mechanism for performing a search. While this location procedure is a difficult problem in the context of the Internet, its implementation on an internal server or group of servers should be simple and efficient.

Several months were spent at Intel exploring the possibility of utilizing the Web to make relevant internal information available to the semiconductor processing research community Portland Technology Development (PTD). This project was undertaken during the period from June to August, 1995. At this time, large companies such as Intel were just beginning to explore the possibility of using the Web as an important tool.

2.1 Unstructured Information

The first issue encountered during this assignment at Intel was the necessity of raising awareness of the availability and potential of the Web. In the process of doing this, we spoke to many individuals about what types of information should be maintained on an internal Web Server. Some common themes became apparent. People concurred that a lot of information floated around the community via email and was stored by hundreds of people for later access. Several engineers and technicians complained that they occa-

sionally discarded documents which seemed irrelevant to their work but which they later desired and could only obtain with effort. Clearly, a single and well-known access point would conserve disk space, as well as assure that everyone had "permanent" access to the same, accurate information.

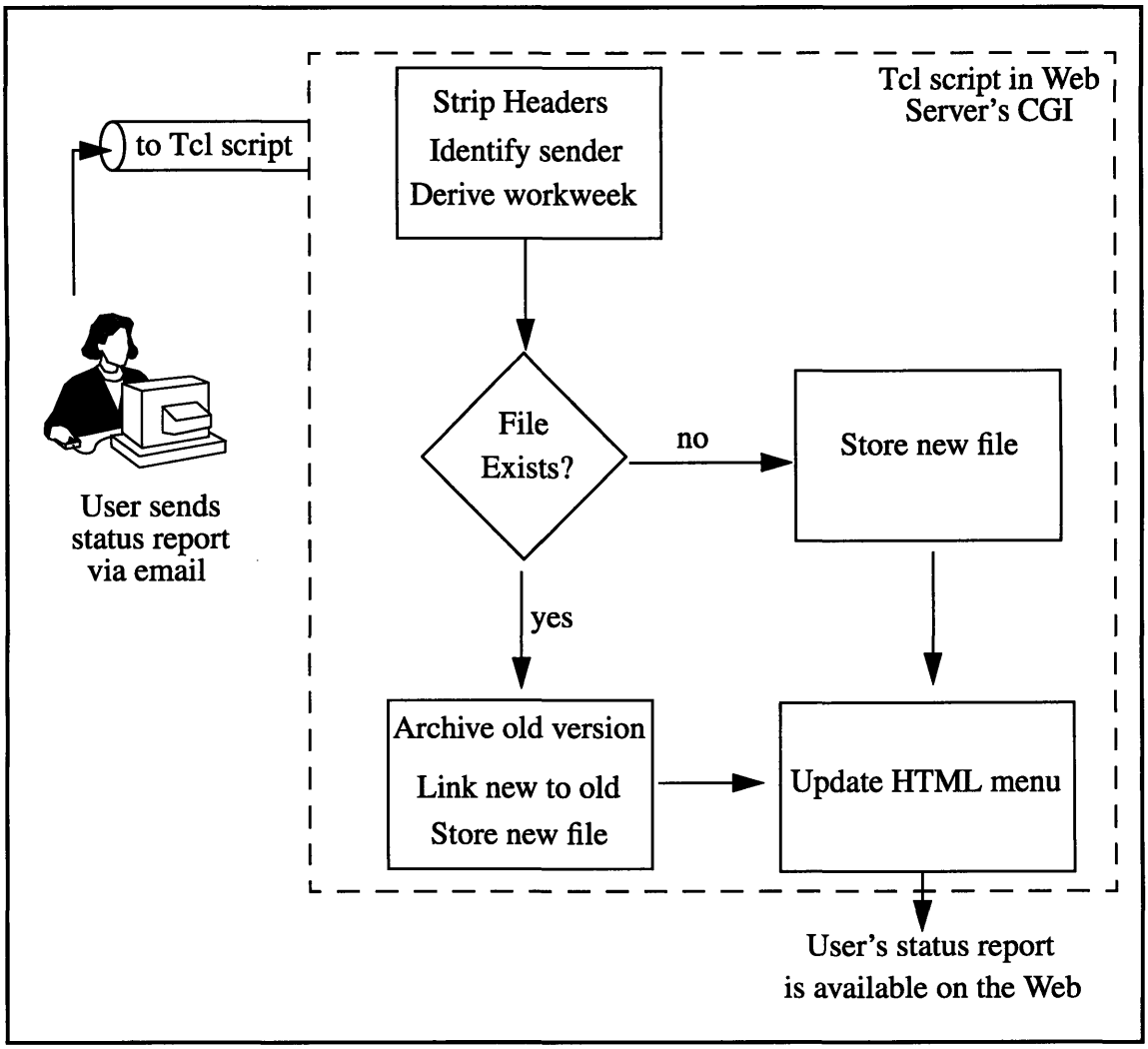
People suggested inclusion of such unstructured and miscellaneous information as departmental or administrative updates, weekly status reports of individuals and groups, white papers, and fabrication machine and lot status. Weekly status reports were of particular interest because of their current handling. At the time, each individual sent his status report to his supervisor and group. A supervisor summarized the group's reports and sent that report up the chain. Finally, a top level report was made available to the entire research organization. Several engineers noted that availability of the status reports of everyone in the hierarchy would be useful since under the current scheme it was difficult to obtain detailed information about progress in groups other than your own.

In developing a mechanism for creating and maintaining an organized repository of status reports on the Web, it was important to consider the target users. The Web was still a fairly new tool, and since browsers were not yet available on the LAN, many of the engineers were not familiar with it. While many of the engineers were interested in learning about the Web and using a Web browser, they were too busy to consider taking responsibility for exploring this untried technology by maintaining a group Web site. It was necessary to implement a system which provided a simple and familiar mechanism for the user to make a status report available on the Web. Since people were accustomed to submitting their status reports to their superior via email, we decided to implement a prototype system that would allow users to email their weekly status reports to the system.

The initial implementation served just one group. An email account was set up on a UNIX machine such that email messages received by that account were piped to a Tcl script. The script separated the headers from the body of the message and wrote the information to a file which could be accessed by a Web browser. If such a file already existed, it was assumed to be a previous report. The old report was moved to an archive directory, and a hyperlink to "Last Week's Report" was added to the foot of the new document. In order to properly identify and store the report for retrieval, information such as the person's name and the work-week of the report needed to be known. To make the submission

process completely transparent to the sender, a table could be maintained which mapped email addresses to names, and the work week could be computed based on the time stamp of the email. A flowchart of this process is shown in Figure 2.1.

Figure 2.1 Flowchart of Status Report Submission Process



There are a couple of problems with this plan. First, the vision of the system is to provide a repository containing the status reports of everyone in the organization. Maintaining a lookup table pairing addresses and names of hundreds of people might be tedious, especially given that most people in the organization frequently used two (cc:Mail and vaxmail) and occasionally three different email accounts. However, it would be possible to do this. Deriving the work week from the timestamp on the email is fairly straight-

forward, assuming that no one ever tries to send a report for a particular week outside of some specified time frame. Finally, as this system expands to include many small working groups, some level of categorization is necessary.

This organization was implemented as a short series of Web pages providing links to increasingly smaller groups, until finally a user reaches a page with a list of individuals and hyperlinks to each one's status report. To do this, the system must know the appropriate branches of the information tree for each incoming report. It was determined that at the beginning of the body of a submission, the system would expect to find some keyword/ value pairs from which it could determine where to place the document in the Web server's directory structure. Some people suggested that this requirement might be enough to deter some potential contributors. Two alternatives were apparent. First, the lookup table could be further expanded to include the hierarchical information associated with each person, but this information is far from static and would require frequent updates and changes. The second idea was to use the Web capability of fill-out forms as an alternative submission mechanism.

The Hyper-Text Markup Language (HTML) used to create Web pages provides rudimentary user interface elements including buttons, pull-down menus, and text input areas. An author may utilize these to create a fill-out form within a Web page. It is important to note that the page is static. When the user clicks on a form's "Submit" button or otherwise activates the process, the browser will package the information entered into the form and send it back to the server with a request for a new page. The URL associated with this request will be a call to a script or program in the Server's Common Gateway Interface. In this case, we created a form with several text input areas for the information about the report, and a large text input area into which we expected the user to paste her status report. An action button sent this information to our standard script for processing.

It is clear that any of these methods of document submission do indeed require an extra step of the submitter than usual, but we hoped that interest in using the Web and putting one's information "on the Web" would be a motivating force. We did make an effort to implement submission mechanisms which did not require that the user master new systems or skills. Among trial users interested in Web technology, the system was popular and well-used. It provided a repository and history of documents, some of which had pre-

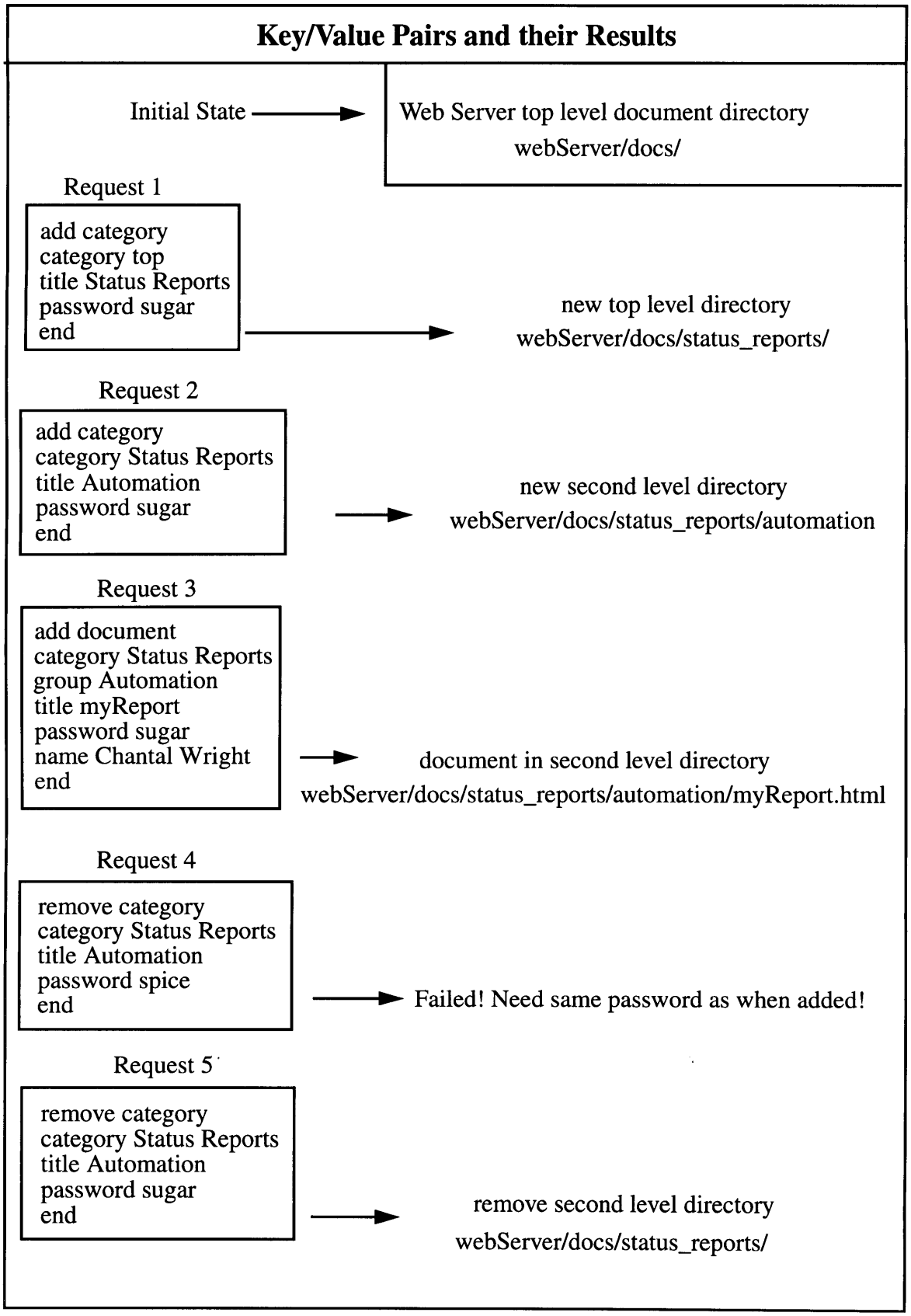
viously not been easily accessible to an appropriately wide audience.

The goal remained to provide a mechanism for adding arbitrary information and documentation to a large Web-accessible repository without requiring frequent system maintenance. The extensive list of documents and information types that was suggested for inclusion in the project made it clear that it would not be possible to predict and therefore predefine locations and addresses for every type of document that would be added. Because there would be no human maintainer to continuously modify the directory structures, we determined that this type of change or addition should also be possible via email or through a form interface on the Web.

The system was expanded to allow users to create new categories of documents when they wished to add a new information type to the hierarchy. The Tcl script was modified to handle more keyword/value pairs, allowing a user to "add" or "remove" a category or a document. Addition of a category simply created a named directory on the Web server in the hierarchy mirrored by the accessible information tree, a series of increasingly specific menu Web pages. This action does require more knowledge of the user who wishes to modify the menus, but those with great interest in expanding and using the system were willing to spend time developing the structure. Providing a form for making this type of change simplifies the procedure, but introduces the risk that too many changes are made by too many people and the structure becomes disorganized, counterintuitive, and repetitive.

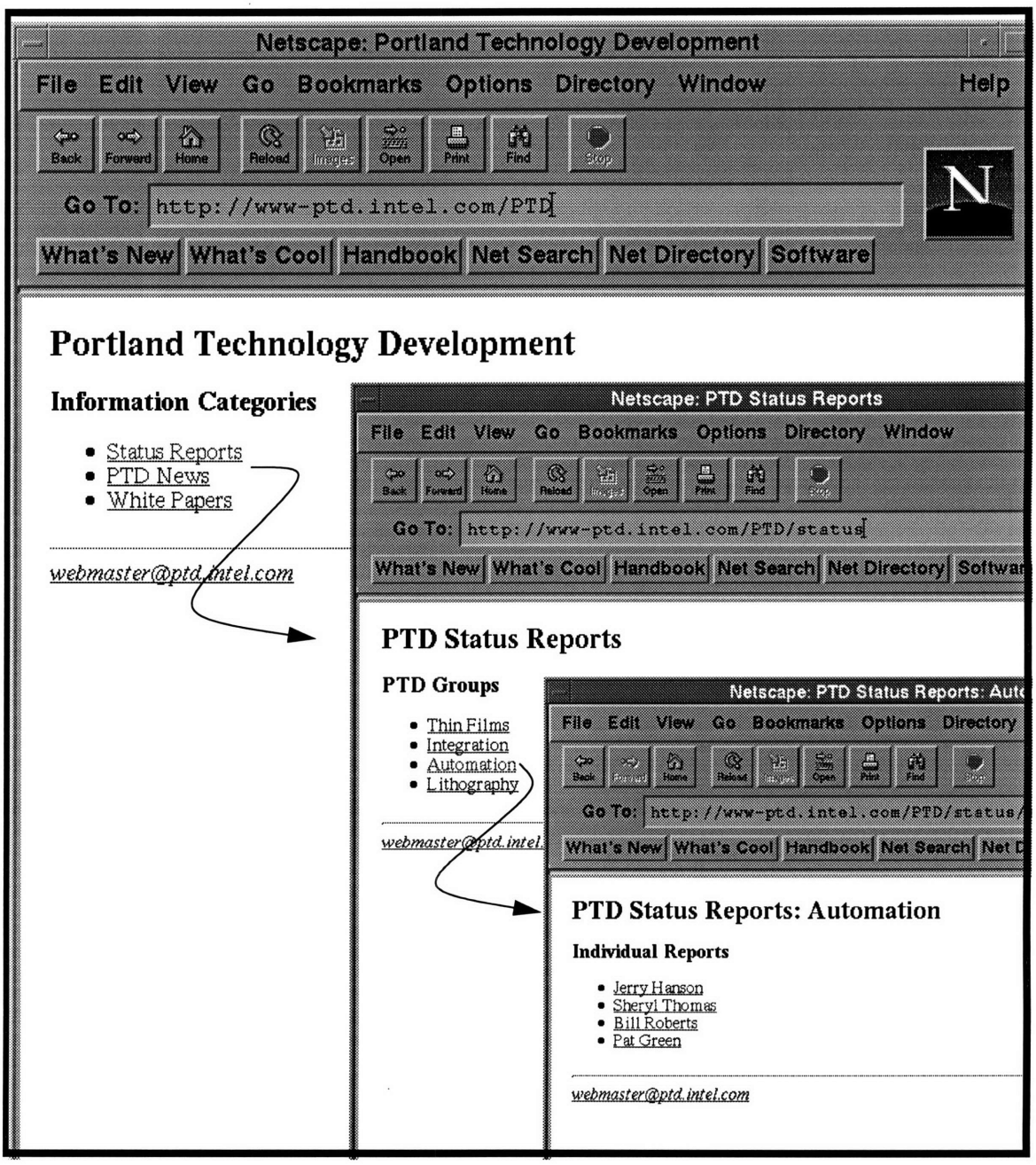
It was determined that some mechanism was necessary to prevent different people from adding, removing, or overwriting a category or document put in position by another. We decided to use a non-robust password system whereby the user would provide this as one of the required keyword/value pairs. Figure 2.2 provides examples of the required keywords and values a user might submit in the process of developing the structure and content of the information repository.

Figure 2.2 Example of Repository Growth



Finally we had a prototype system which could evolve into a structured repository of information. The tree-like sequence of menu pages would provide a simple way to locate information. Figure 2.3 displays an example of this structure. As new versions of

Figure 2.3 A sequence of HTML pages in the repository



documents arrived, old copies were archived so that information would remain available

for some time period. This prototype system was evaluated and seen to have great potential for providing easy and consistent access to information, assuring a single "current" copy of the information, and minimizing the disk space allocated to multiple copies of the same information.

Of course, there are some impediments to such a system becoming the primary source of the information it contains. Foremost was the need to introduce the Web as a common tool in the research environment. Many were altogether unfamiliar with the technology, while others were not yet considering it in the context of internal information flow. Another concern is security. An important restriction is made on information placed on the Web, because although the Server is not accessible outside of Intel, any documentation contained therein can be easily captured and transmitted. The restriction, then, is simply that documents which were not generally electronically available could not be made available on the Web. As acceptance of the Web as a useful tool grows, a system such as the one we have prototyped could be an important improvement in communication and information distribution.

2.2 A System for Document Revision and Ratification

In a fabrication environment, there are several documents which provide fundamental instructions or descriptions available to everyone. That a document is widely distributed suggests that it might prove useful to maintain it on the Web. We decided to explore the possibility in the context of a large document with many contributing authors.

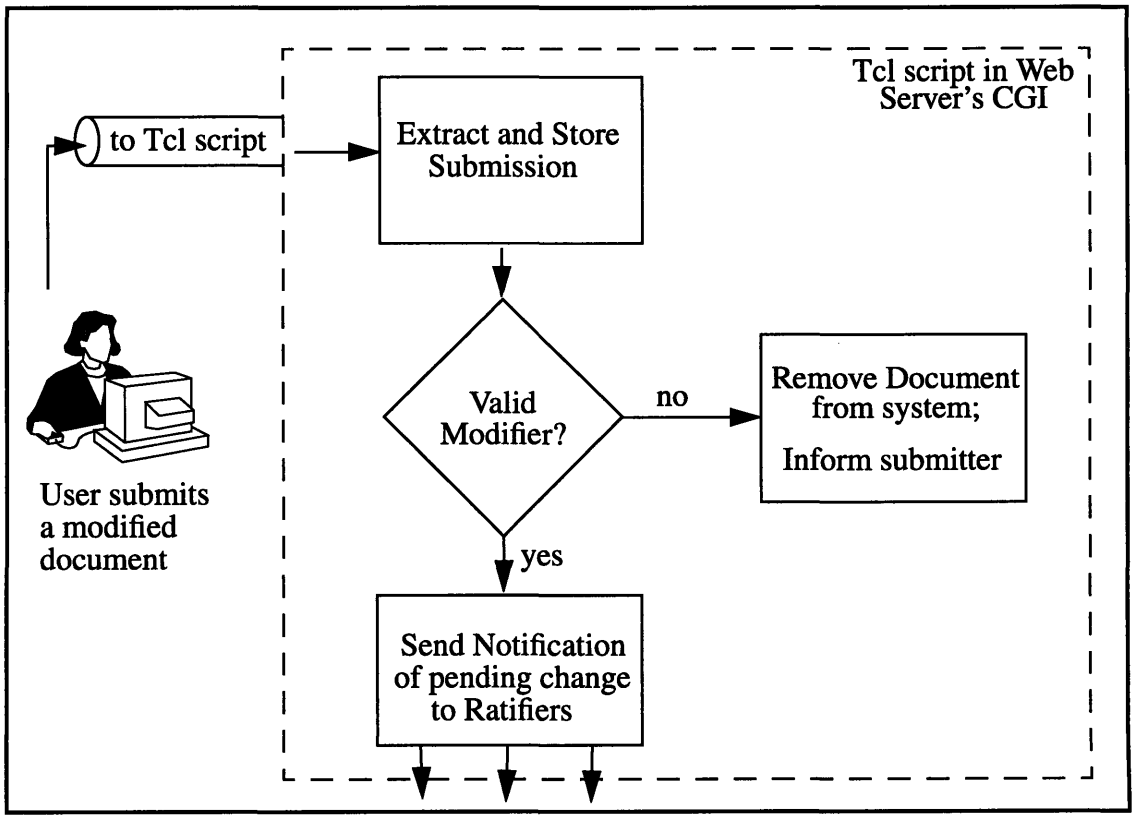
A prototype system was specified and implemented which was intended to facilitate the process of revising and accepting revisions of important documents. The document that we selected as a model for this process is composed of many different sections. Each section is generally maintained and updated by a particular set of individuals. When a change is made by one of the authorized modifiers, this proposed version of the section circulates among a defined group of ratifiers. If all of these individuals approve the change, it replaces that segment in the larger active document.

The goal was to use the Web to automate this process so that it occurred entirely on-line. The model document used to prototype this system was composed of various

Microsoft Word, Excel, and PowerPoint files, the formats of most documents at Intel. Instead of employing one of the currently existing and very rudimentary programs for converting these formats to HTML, we decided to leave them in their native format and use helper applications to launch the appropriate viewers from the Web browser. One disadvantage of this method is that there is not a continuous document available for perusal; instead, there is a sort of table of contents page which has links to each of the separate files which make up the total document. However, this does simplify the process of editing and replacing segments of the document, which is its normal mode of evolution.

The entire document was placed on the Web in the format described above. The system allowed the author to submit a new version of one of the document's files via email as a cc:Mail attachment. Email received at this special address was piped to a Tcl script which separated the attachment from the email and stored it in a special directory where it would await ratification or rejection. A flowchart describing this process is shown in Figure 2.4.

Figure 2.4 Flowchart for Submission of a Modified Document



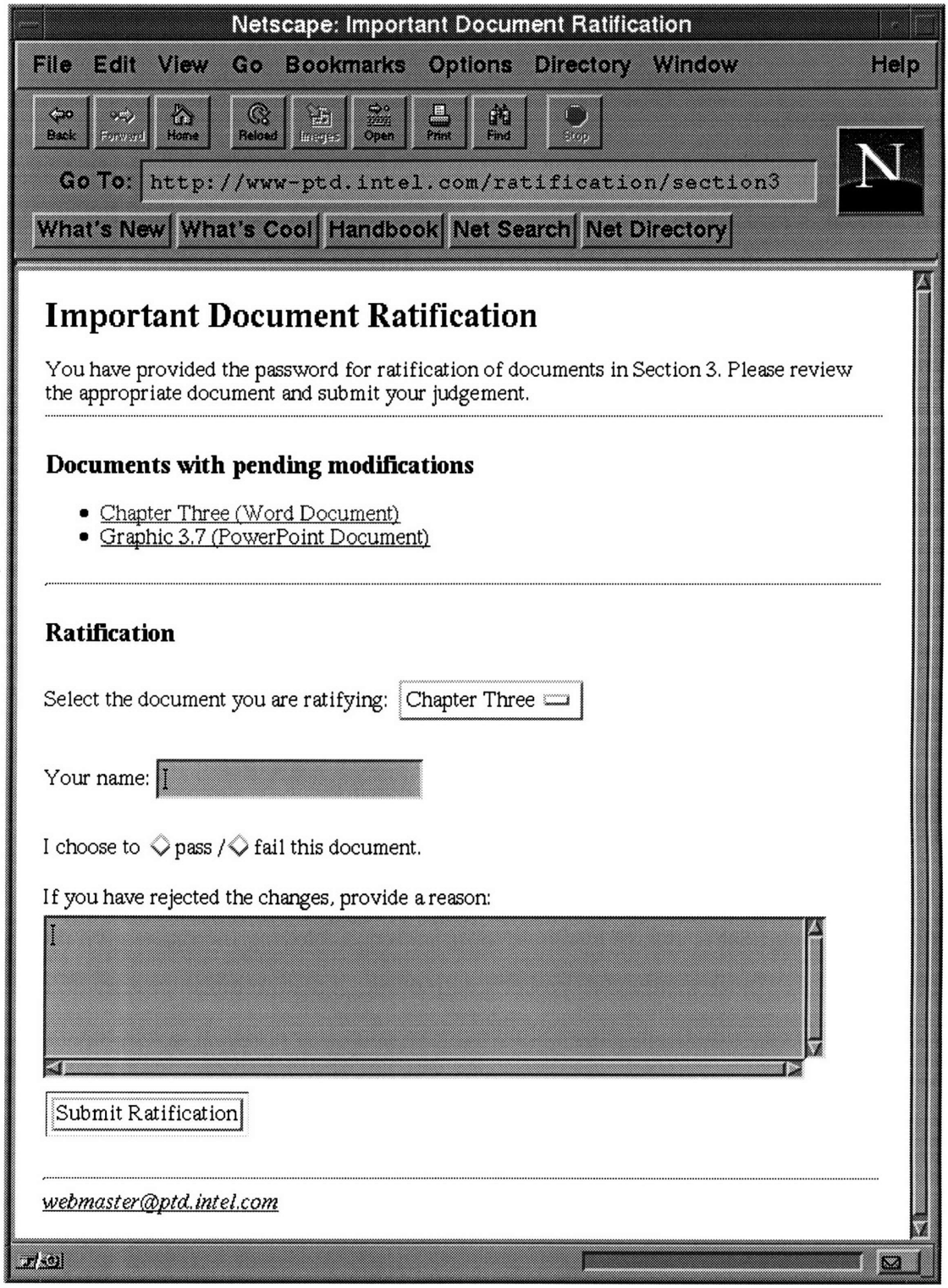
The script also verified that the sender of the modification was indeed a valid modifier of the particular document received. If not, the file would be removed from the system and a message sent to the submitter indicating the reason for no further processing of the submitted document.

If the source was a valid one, the system delivered notification of a pending modification to all those required to ratify changes to that particular file. This notification included the name of the author who had submitted the document, the name of the document, and the URL of a Web page supporting the ratification process. This page is shown in Figure 2.5. The ratifiers would then visit the ratification page which contained a link to the new document as well as a form for ratification. Access to this page was protected by password so that only the appropriate persons could view it. The ratifiers would each vote to ratify the changes or to reject them, in which case they would be asked to provide a reason.

If any individual rejected the document, his reasoning and identity was sent by email to all of the ratifiers as well as to the author. The rejected document was then removed from the system and the process could begin again with a new submission. If a document modification was ratified, the document would replace the old version in the appropriate publicly available (within PTD) Web repository.

This system met the specifications and was transported to a Chandler site for installation and evaluation. The system succeeds in assuring that the most recent version of the entire document is always available to its audience. Moving the ratification process to the Web allows those involved to proceed with minimal distraction since they do not initially have to coordinate any physical meeting or transfer. Further, it is assured that no mistakes are made in effecting or communicating the final results of document change.

Figure 2.5 On-line Document Ratification



2.3 Conclusion

These prototypes demonstrate an attempt to organize and present the massive amounts of disconnected and unstructured information associated with process technology development. We have demonstrated that the Web can be successfully employed to present this type of static and regularly updated documentation. This information is crucial to research efforts; facilitation of its presentation and distribution complements a similar objective in manipulating structured and dynamic processing information.

Chapter 3

CAFE Database Traversal

A key step in the development and growth of distributed research is dynamic data sharing. In the semiconductor field, data describing laboratory processing is critical. Users must be able to monitor current experiments in order to take advantage of and build upon the work performed at remote sites. The Microsystems Technology Laboratories (MTL) at MIT maintains its processing information in a local database; making this information readily available to coresearchers will promote distributed research. The Web and its CGI facility provide support for implementing this type of a system. We have developed a system which allows a user to employ a Web client to browse and traverse the MTL database objects and related information.

3.1 The Problem

At MTL, information about process activity is maintained in an INGRES database with a GESTALT database interface [Hey89]. The menu driven software (Computer-Aided Fabrication Environment, CAFE) which manipulates the data provides a bidirectional interface between process design and the fabrication facility [McI92]. The CAFE database maintains current information describing objects whose interaction represents

the fabrication process. These objects include users, lots, machines, facilities, processes, and wafers. Users within MTL frequently access and utilize information in the database as they develop experiments and analyze results. We would like to extend this capability to a larger community of researchers.

The data representation employed by CAFE application programs is the Process Flow Representation (PFR), a format created in previous research at MIT and used extensively in MTL [McI90]. The PFR was designed to support local integrated application programs; the specificity of the associated programmatic interface does not lend itself to human interaction and distributed research. However, providing PFR data to remote researchers in a related textual format can contribute to a distributed research environment, if only for casual perusal.

Given the ultimate goal of sharing information to advance research throughout the academic community, it is clear that a standard data representation must be implemented and employed by all of the participants. Until this commonality is realized, we are limited to sharing information at a more informal level. In this chapter, we describe mechanisms for such sharing of process information; in Chapter 4, we present work toward the goal of distributed process design using formal process representations.

3.2 The Solution

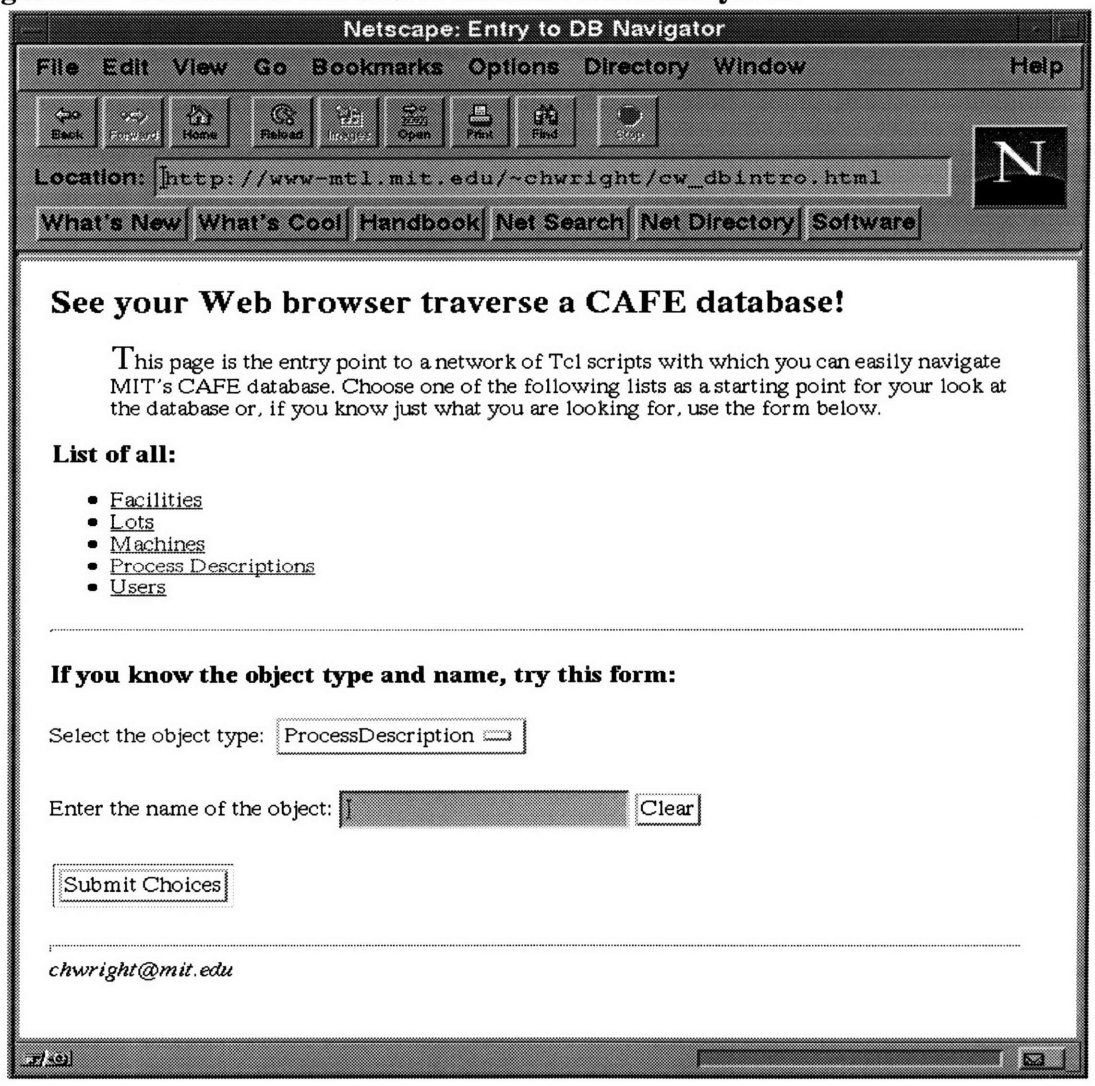
We wish to allow remote researchers to effectively browse and traverse the CAFE database using a Web client. Because the information maintained by CAFE is updated as wafers are processed in the lab, a static preparation and presentation of information is not sufficient. Using the semi-dynamic provision of the Web (the Common Gateway Interface), we implemented a series of Tcl scripts which enable traversal of the CAFE database. The introductory page of the system is shown in Figure 3.1. This system builds upon an early prototype of a Web interface to CAFE process description objects [Pri94].

The scripts use some simple routines to access the CAFE database and retrieve information on the fly [Bon93]. As a user traverses the data, she may find herself on a page describing a process step, including an attribute called "machine." Such attributes are presented as hyperlinks to other objects in the database such as, in this example, a machine.

Sending a request for an object by activating a hyperlink provides the Server with the information it needs to query the database. The querying script returns standard output in the form of HTML which is delivered to the client and presented to the user. Such a "page," describing a process flow object, is shown in Figure 3.2.

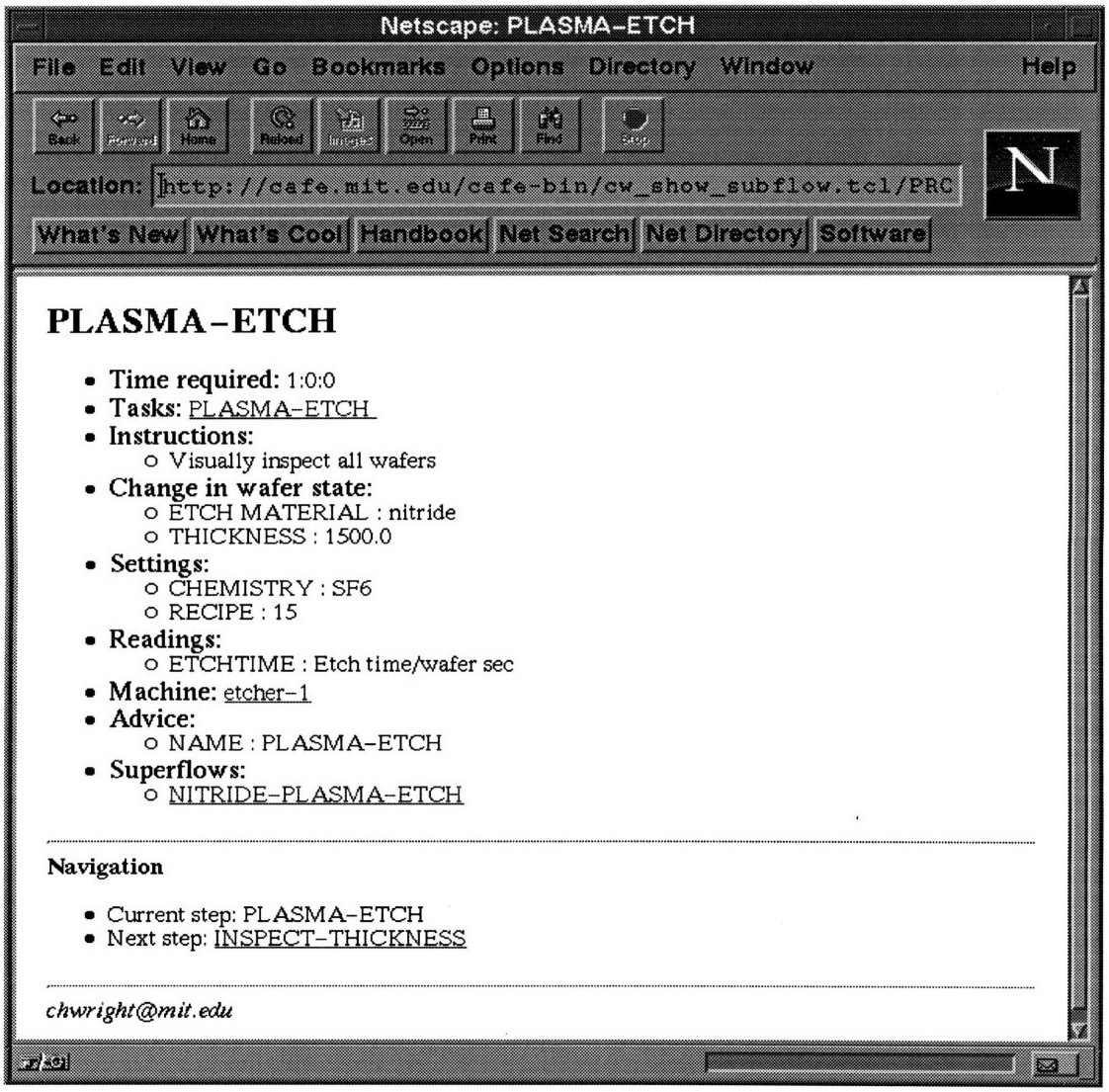
The ability to view the data contained within CAFE is not new. However, using the Web for this purpose does provide certain advantages, including a simple and uniform interface for all users whether at MIT or a remote site. The straightforward mapping of database object interaction to hyperlinks also motivates the use of the Web. Another significant advantage of hyperlinks in such a system is the capability to include access to relevant process information not explicitly contained in the database.

Figure 3.1 Interface to the CAFE database traversal system



PFR flow definition files, for example, are not contained within the database although a reference to them is included as an attribute of a process description. It is a simple step to modify the Tcl scripts which generate the HTML view of the process description objects to include a hyperlink to these files. This type of addition to the system greatly enhances its power. Links can be made to virtually any data format; thus we may link objects to a wide range of associated information, including pictures and graphics, and potentially even machine interfaces.

Figure 3.2 A Process Flow Object



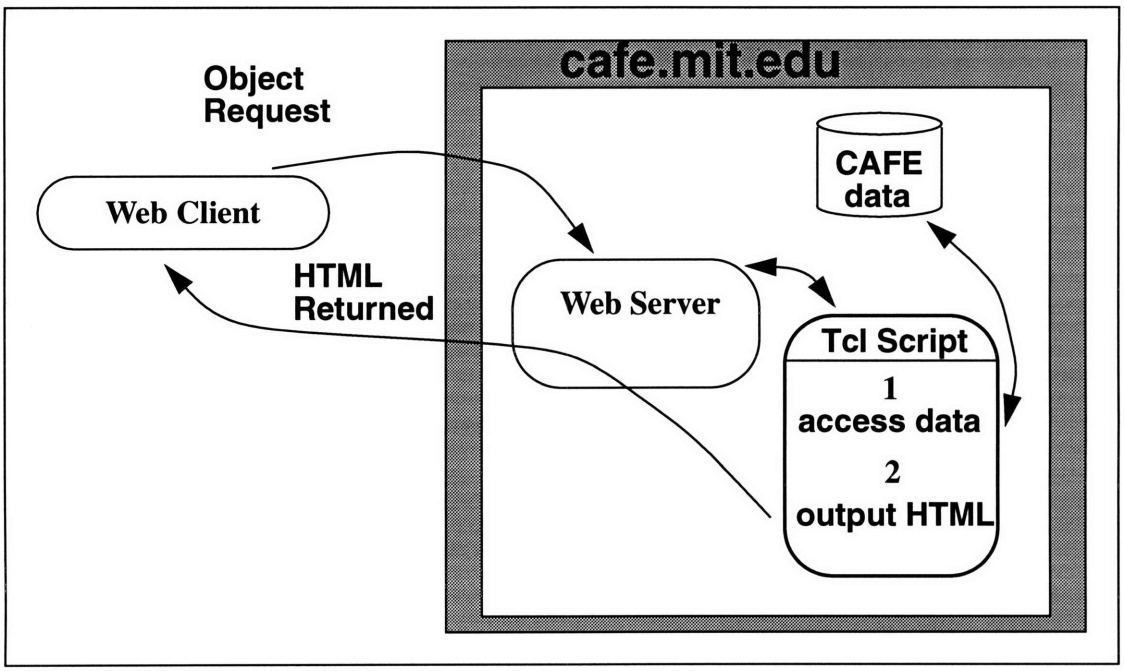
By exploiting the capabilities of hyperlinks, we can create an increasingly simple

mechanism for the user to explore a diverse set of information related to a given object. We may connect arbitrary information sources at a single well-known access point on the Web, thereby providing researchers, both local and remote, with a consistent and simple means to explore data and related resources.

3.3 The Implementation

As described in Section 1.1, a Web server's Common Gateway Interface defines a standard for external gateway programs to interface with HTTP servers. We have taken advantage of this capability in producing the system described above. The server recognizes a URL as an executable script based on a list of file extensions in the server initialization and setup files; our server knows that files with the extension ".tcl" residing in a particular directory will be executed on the server rather than returned immediately to the client (as a standard HTML file would be). Each time any Web client requests the URL corresponding to one of the CGI scripts, the server executes it in real-time. The standard output of the script is passed by the server back to the requesting client, and the client treats that input stream as a normal HTML page. Specifically, we employ this mechanism to accept and satisfy requests for objects within the CAFE database. The operation just described is depicted in Figure 3.3.

Figure 3.3 Basic CGI operation



The CGI also defines a mechanism whereby the client may send arbitrary parameters to a script. Such information is delivered to the server by appending extra information to the URL. The server places this string in its PATH_INFO environment variable where it can be accessed by the script. Our system expects an object identification string to be delivered in this fashion each time any script is invoked.

For each class of objects, there is a script to retrieve the associated set of PFR attributes, given the instantiation identification string. The result is a number of very similar scripts which differ only in the set of attributes they query. While this may seem redundant, it minimizes the time a user must wait to view the requested object since no time is wasted querying the database for nonexistent attributes.

Simple C and Tcl routines for performing queries on our particular database exist and are extensively employed by these scripts [Bon93]. As the attributes are collected, an HTML page describing the object is written to standard output, and, thus, to the Web client. A sample script which retrieves two attributes of a process flow object is shown in Figure 3.4.

Figure 3.4 Sample Script for process object

```
#!/usr/cafe/bin/tclCAFE -f

puts stdout "Content-type: text/HTML\n\n"

set id_parts [split $env(PATH_INFO) /]
set objID "GDM#[lindex $id_parts 1]#[lindex $id_parts 2]"
set name [dbeval select name $objID]

set time [dbeval select timerequired $objID]
set machine [dbeval select machine $objID]

puts stdout "<title>$name</title>"
puts stdout "<h2>$name</h2>"
puts stdout "<ul>"
puts stdout "<li><b>Time Required:</b> $time"
puts stdout "<li><b>Machine: </b><a href=/cafe-bin/show_machine.tcl/$machine>
            [dbeval select name $machine]</a>"

puts stdout "</ul><hr><i>chwright@mit.edu</i>"
```

Write the required header info to the browser

Get parameters from the PATH_INFO environment variable

Get process object attributes from the database

Write HTML to the browser

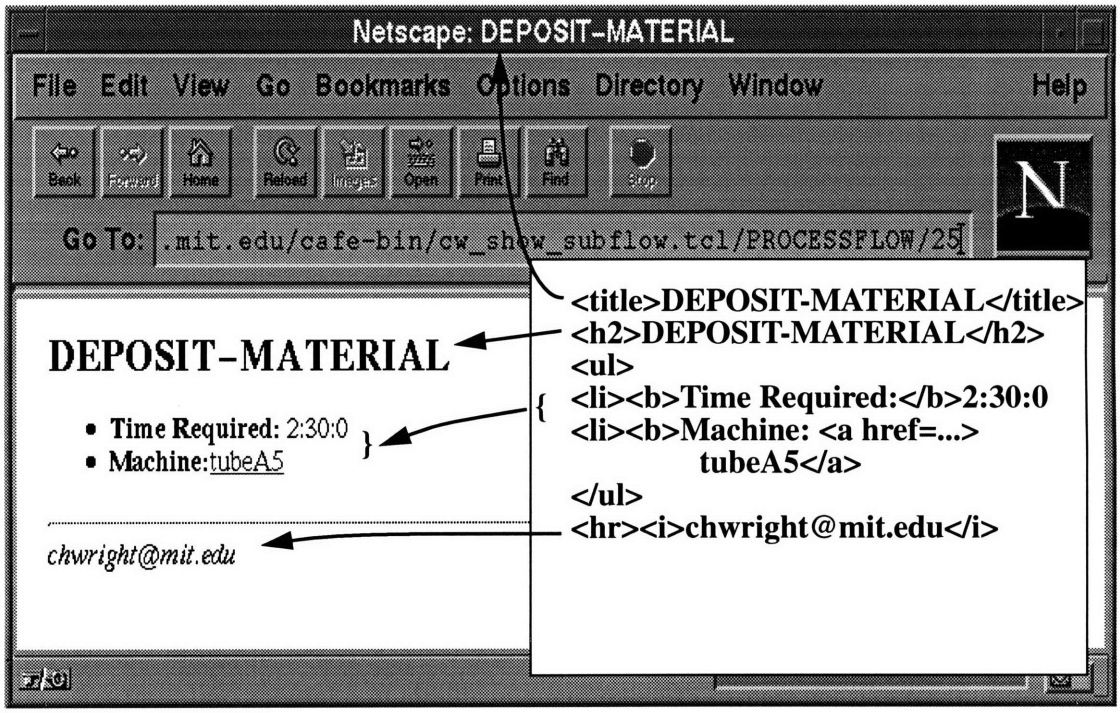
A URL we might request to invoke this script is

http://cafe.mit.edu/cafe-bin/cw_show_subflow.tcl/PROCESSFLOW/25

In examining this URL, we note first that we are making a request of the machine at MIT called cafe. Next, we note that "cafe-bin" is the directory associated with executing CGI programs. We see that the script is called "cw_show_subflow.tcl." The string "/PROCESS-FLOW/25" is the value stored by the server in the PATH_INFO environment variable.

Upon execution, the first action of the script is to respond to the Web client by sending the necessary header describing the type of data to follow. Next, the script constructs the PFR process identification string based on the specific information transmitted in the URL; in this case, the resulting string is "GDM#PROCESSFLOW#25." We use a series of locally defined "dbeval" commands to query the database for the attributes name, timerequired, and machine associated with the process object. Finally, the information is written to standard out as HTML for presentation at the client. Figure 3.5 shows the Web page which would result from calling the URL above. The corresponding HTML is shown exactly as it would be delivered from the script.

Figure 3.5 Sample Result of Script Execution



3.4 Conclusion

In this chapter, we have described a mechanism by which MTL process and related information (e.g. machine, facility, and task data) can be made available to coresearchers via a Web browser. This level of simple information sharing is an important step toward distributed research. The core capabilities of the Web have thus allowed us to provide both the static document display described in Chapter 2, and distributed access to dynamic information. In Chapter 4, we will consider how the Web and supporting technologies such as the Java language can be employed to realize distributed utilization of structured information in process development and design.

Chapter 4

A Distributed Process Flow Editor

Chapter 3 described a mechanism whereby information about process flows and processing in the MTL laboratories can be made available to a larger community of researchers. While this increased information availability is an important step toward distributed research, it was noted that a common and standard data representation, particularly of process information, is an absolute necessity to that end. With a standard data representation, the building blocks of process descriptions could be shared among researchers for immediate incorporation and use in both fabrication and simulation.

At MTL, process flow descriptions are specified in a format consistent with its unique Process Flow Representation (PFR). Process flow design has been facilitated by the implementation of graphical process flow editors. A user may thus build a flow graphically without knowledge of the underlying data representation. These editors are limited to use in MTL or other facilities running the CAFE system by two factors; first, they manipulate only PFR data, and second, even if the data representation were standard, these Tcl/Tk implementations reside on MTL (or local facility) computer systems and are not conveniently shared with remote sites and users.

The first problem will be overcome when a standard data representation is employed by researchers throughout the field. In fact, a standard has been developed by academic and industrial committees but has yet to be widely implemented. This Semiconductor Process Representation (SPR) and its status will be discussed in this chapter.

The second problem can potentially be alleviated by implementing the software in Java instead of Tcl/Tk. This way, a program could be accessed by any number of remote researchers without requiring specialized installation and management at different sites. This chapter will discuss a Java implementation of a process flow editor. This editor attempts to maintain and represent data in the SPR format, but a thorough PFR to SPR conversion is not the goal of this thesis. Instead, we demonstrate the potential use of Java for distributed semiconductor process development with a cursory translation of PFR to SPR and an editor for processes in an SPR process repository. The MTL CAFE database will be used as our example process repository.

If researchers maintained information in a common format, it would be a simple extension to have this distributable Java process flow editor access and draw upon multiple SPR process flow repositories. A developer could build a flow combining steps from a variety of remote sources. Distributed research in process/device development and optimization can benefit greatly from this increased level of information sharing.

4.1 Semiconductor Process Representation and Distributed Research

In large companies, distributed research and development is standard procedure; this is possible only because the participants have agreed upon a common information representation. In academia, different institutions have historically employed uniquely developed and specialized representations. Now that distributed research among remote academic facilities has become feasible and important to progress in the field, attention has been turned to the development and implementation of a standard data representation.

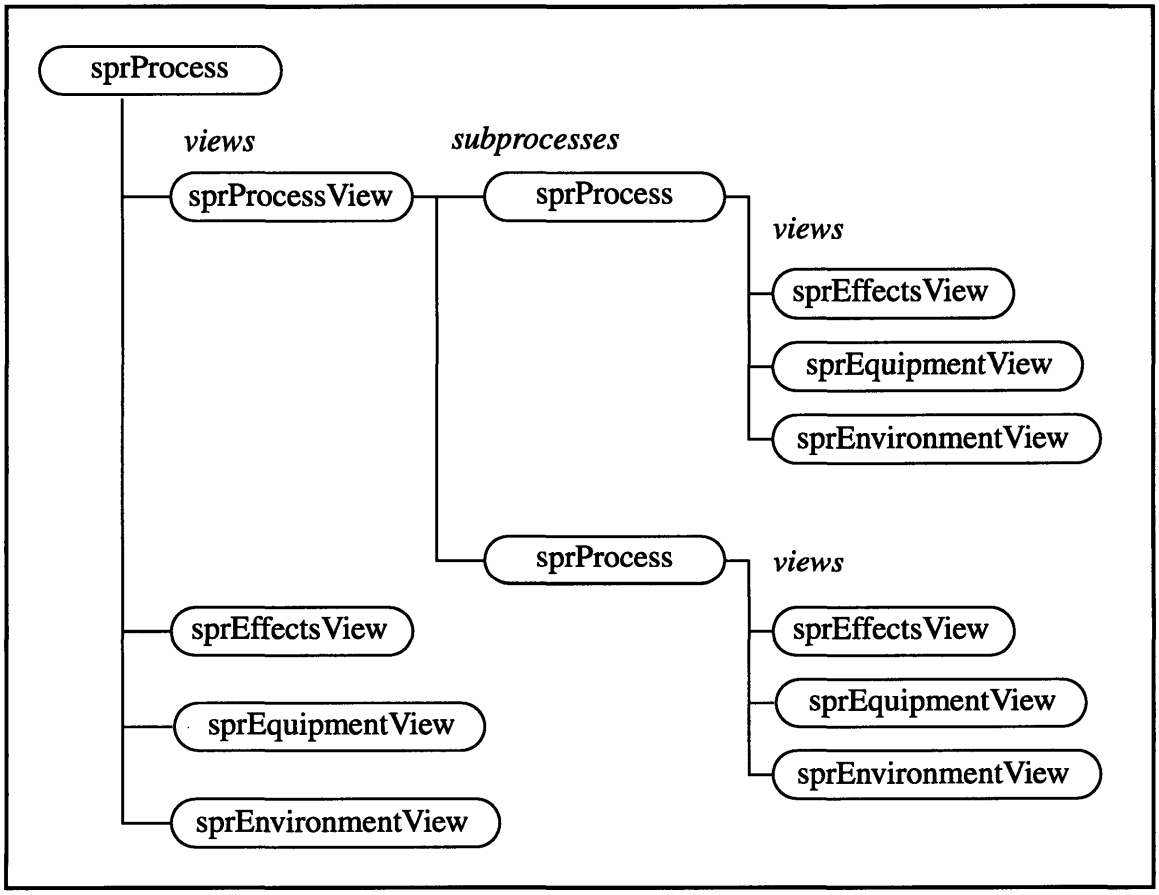
Based on research at MIT, Stanford, and elsewhere, the Semiconductor Process Representation (SPR) has been developed to provide a standard way to communicate information about fabrication processes. Systems and researchers in both process design and manufacturing can use SPR to immediately (without translation) share and access process information.

The two components of the SPR are the programming interface and the information model. A standard programming interface is necessary so that distinct software tools and systems can access and manipulate the same process information. We will be con-

cerned primarily with the standard information model. The information model provides an agreed upon way for people and systems to organize and represent process data.

Fabrication information is organized by the SPR information model into a number of perspectives, or "views." A process can generally be broken down into increasingly small substages; this perspective of a step is its process view. At each level of detail in the process view, the information associated with that step is typically broken into three additional views: the equipment view, the environment view, and the effects view. A diagram of this structure is provided in Figure 4.1.

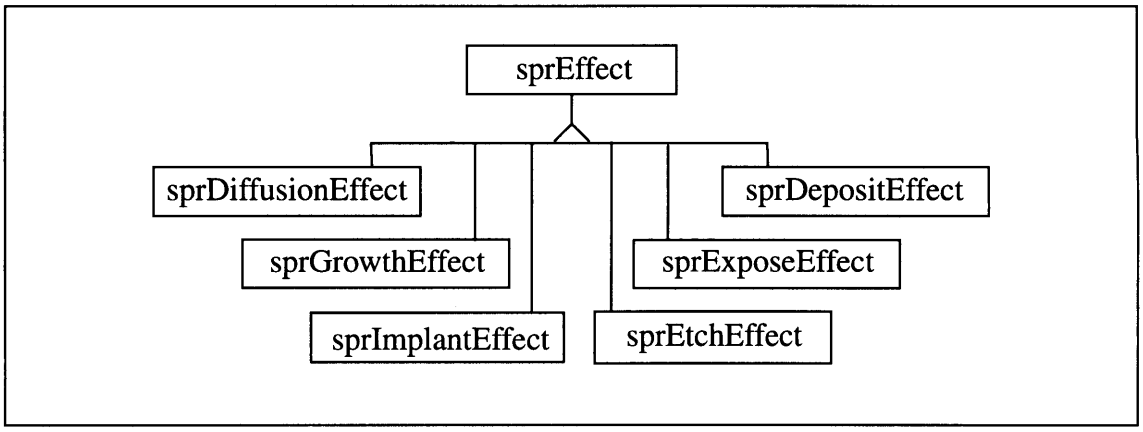
Figure 4.1 A process step and its associated views



Each view is subsequently decomposed into smaller units of information. These view elements have different meanings for each of the different views. Figure 4.1 depicts sequential sprProcesses as the elements composing an sprProcessView. The sprEffects-View consists of some number of sprEffects which describe changes to the state of the

wafer. Figure 4.2 shows some standard sprEffects. Similarly, the sprEnvironmentView is made up of some number of sprEnvironments (such as sprGasEnvironment), and the sprEquipmentView is composed of sprEquipmentStates.

Figure 4.2 View elements of the sprEffectsView



Each view element is completely described by sprParameters, which are name/value pairs. In the three view element types just described, the sprParameters can be thought of as state variables describing some aspect of the wafer, environment, or equipment, respectively, during some specified interval of time.

To this point, the universities associated with the development of the SPR have continued to use their individual representations; thus, the SPR has not yet been thoroughly implemented. As stated above, we have not attempted to perform the initial exhaustive implementation. However, in developing a process design editing tool, we have prototyped the SPR information model in the Java language, and we perform a conversion of PFR to SPR so that we may manage and present semiconductor information (extracted from the PFR-based CAFE process repository) in the SPR format.

4.2 A Java Implementation of a Process Flow Editor

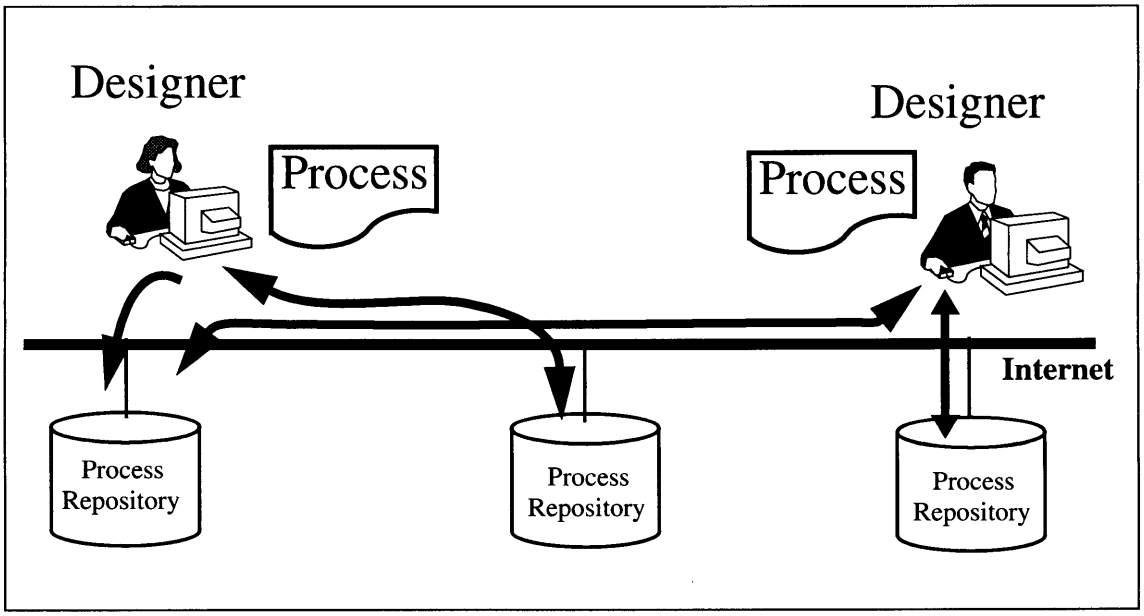
A functional process flow editor allows users to build, retrieve, view, and edit process flows. Such an editor written in Java does not require any specialized software installation of a user, but will be transported across the Web and thus instantly available to remote users. The functionality and graphical presentation of our process editor has been

patterned after tools previously implemented for use in MTL in Tcl/Tk by Albert Woo [Woo93], Greg Fischer, and Mike McIlrath; a key design difference from these versions is our attempt to change the focus of data representation to the SPR.

4.2.1 Design and Prototype

The vision for a distributed process editor includes enabling the user to access a number of remote process repositories. An editor with a well-defined interface for receiving SPR data objects could be connected to any repository which conformed to the specifications. Thus, we can imagine that any institution could implement a translation from its local data representation to the SPR and then package SPR objects for use with this ideal process editor. Figure 4.3 illustrates the potential of such a system.

Figure 4.3 Ideal Distributed Process Design



The primary barrier to achieving this ideal is clearly the absence of any serious efforts to implement and use the SPR. That is not the only impediment, however.

As Java and its capabilities are only just being developed and explored, there are strict security restrictions imposed by Web browsers such as Netscape. While Java does support socket communication and file manipulation, a Java applet is not allowed (by Netscape) to fully exploit these capabilities. An applet is allowed either to manipulate files

on the executing host or to interact with the network. If the applet is loaded over the network (i.e., not from a local directory), it has already "interacted" with the network and is hence not allowed to access local files. Furthermore, such an applet is restricted in its network communications to that host from which it came. The two most serious consequences of these restrictions to our plans are first, that a Java process design tool can only access one data repository (unless multiple repositories reside on the same machine), which must exist on the same host as the Java program; and second, that a user cannot perform save and restore functions of a process flow to her own local file system. These constraints force us to prototype a system which may only access local (CAFE) information. It is our hope that emerging security enhancements to Netscape and other browsers will overcome this barrier in the future.

Since the SPR has not been fully implemented, we are not prepared to specify a standard SPR object interface for our tool. We have thus taken some liberty in designing this interface with attention to our specific task of translating PFR to SPR and to the object packaging mechanism we have chosen.

The PFR is a subset of the SPR. Data objects described by the PFR are associated with a number of attributes, some of which map fairly well to the SPR data structure. We have chosen to simply extract these attributes from a selected PFR object and use them to build a corresponding (albeit simple) SPR object. The structure of an SPR object was designed with two sets of guidelines in mind. First, we have attempted to implement an object structure consistent with the specifications of the published SPR standard. The second consideration concerns the network transfer of an object from its source to a remote user's process flow editor. We have chosen to achieve this transfer using an object packaging system developed by John Carney [Car95].

This system allows us to build objects and then "pack" each one into a serialized ASCII buffer for transmission across the network. As defined by this system, an object is simply a collection of slots, each identified by a unique name. In order to use this object template to represent more complex SPR data, we frequently fill a slot in one object with another previously packed object. At the receiving end of a network transmission, Carney provides routines which recreate the object from the ASCII buffer. Carney has implemented his message packaging libraries in C and Tcl; we have ported a skeleton of the sys-

tem to Java. The process design tool expects to receive ASCII buffers which it can unpack into SPR objects. The detailed structure of such an object will be discussed below.

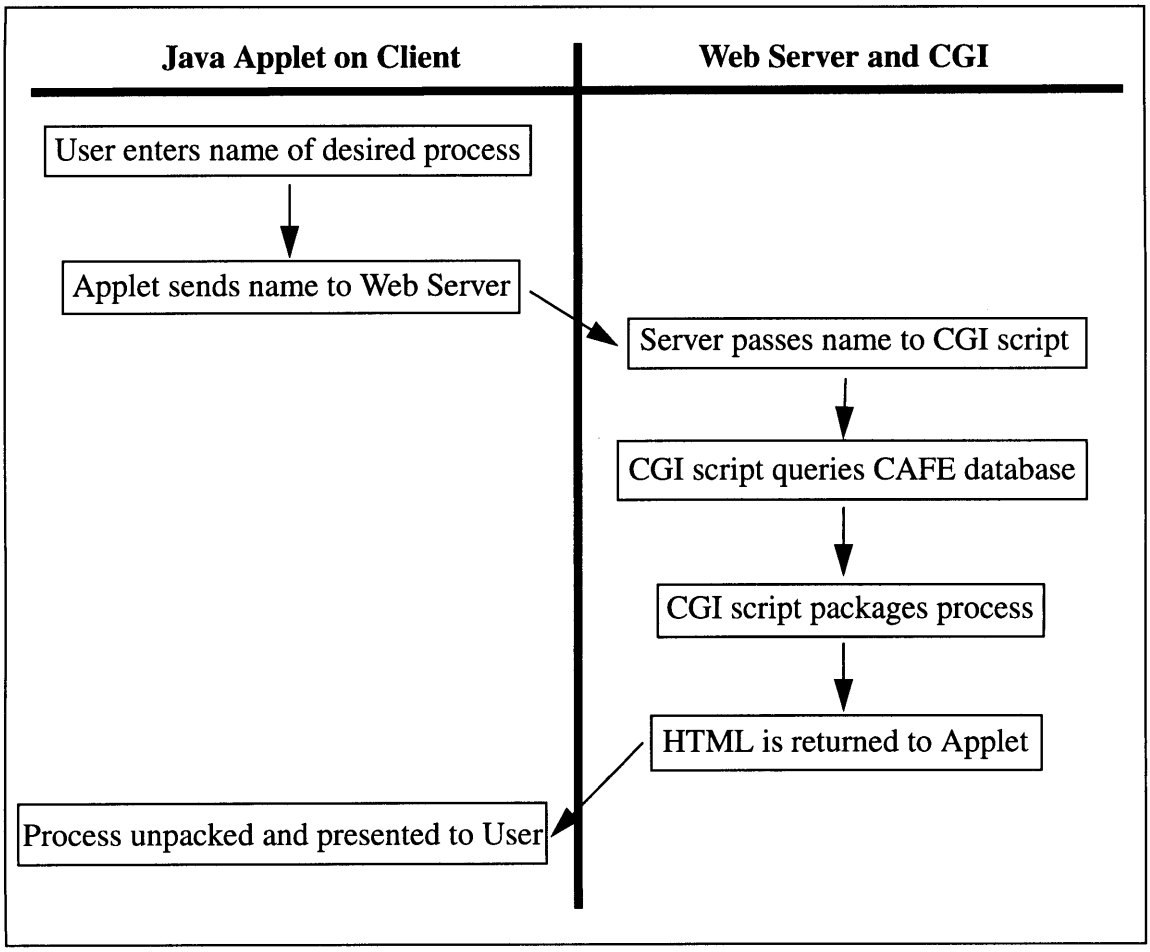
4.2.2 Development of server side

Java does support socket communication by applets; however, rather than establish a daemon on the server machine to respond to process information requests, we have chosen to take advantage of the Common Gateway Interface. We expect the Java applet to make a request of the Web server using the POST method associated with the CGI. This action results in a stream of information being piped to the standard input of the script specified in the requested URL.

We have developed some Tcl scripts to respond to the various requests that the process flow editor may make. The use of the CGI and Tcl allowed us to take advantage of scripts and routines previously developed for accessing data in the CAFE database. The scripts obtain PFR data from the database, reformat it as an SPR object ready for transmission, and write the data to standard out, thereby satisfying the applet's request.

A process designer will frequently wish to retrieve a process flow from the database. The Java tool allows the user to enter the name of the desired process, which may range in complexity from a facility's baseline process to a single processing step. The applet delivers this name to the CGI script as a request for the associated sprProcessView. The Tcl script finds the current version of the named process in the database and packages and transmits it to the editor. This flow is outlined in Figure 4.4. Since each flow is an unknown tree of steps, the script works recursively to pack the sprProcessView. As noted previously in Figure 4.1, the sprProcessView consists of SPR representations of a number of subprocesses. The sprProcessView we transmit to the process editor has this structure, but we have included several other slots of information which are not strictly included in the SPR standard. (However, we could package these extraneous values in an sprEncapsulatedView, an SPR provision for model extensions.)

Figure 4.4 User requests a process by name

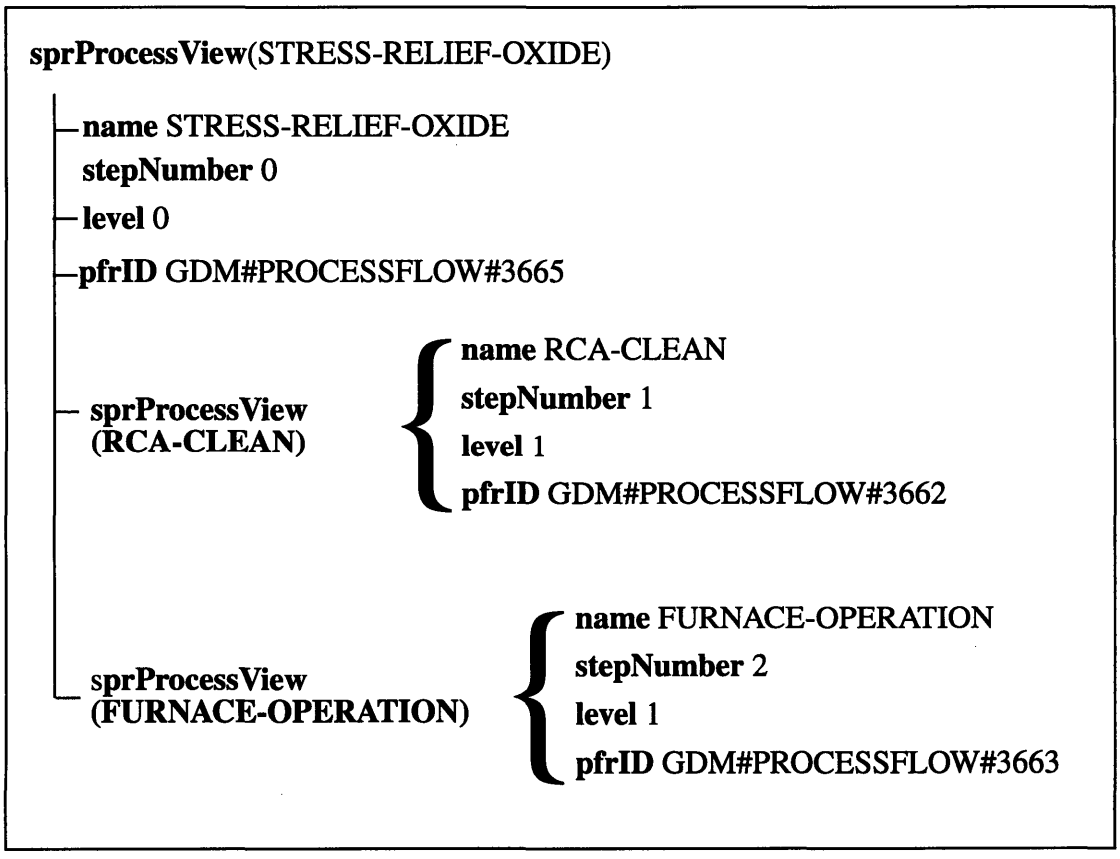


Specifically, we attach three slots to each sprProcessView in our package in addition to the step's name and any substep sprProcessViews. First, we note that the method of recursively packaging and transmitting an object results in a seemingly random ordering of slots in a received process object. Thus, when packaging a process object, we attach to each step in the tree a step number which enables us to easily reconstruct the sequence upon reception. The first node in the tree hierarchy (which bears the process name requested by the user), is always tagged as step zero. Second, instead of extending the sprProcessView to describe the siblings, children, or parents of a process step, we attach to each step a slot labeled level which describes the depth of that step within the tree. The first node in the hierarchy is level zero, its children are level one, and so on throughout the tree. Finally, in a concession to our ties to the CAFE database, we include the PFR identification string for each step. Because the name of a step does not uniquely identify it

within the database, we have decided to pass its database identification string so that a user may later choose to retrieve more of its sprViews. If we defined an interface to accept only strictly SPR information, we would require a complete SPR to PFR translator which maintained some mapping of an SPR object to its PFR counterpart for this purpose.

A diagram of an SPR process object prepared for transmission is shown in Figure 4.5. This example demonstrates the structure of the response to a request for the process STRESS-RELIEF-OXIDE. The sprProcessView of that step is the top of the hierarchy of the object. As all packaged steps, it contains four simple slots which hold the process name, step number, step level, and PFR identification string. STRESS-RELIEF-OXIDE has two subprocesses which have been recursively packaged into slots.

Figure 4.5 sprProcess object packaging for transmission

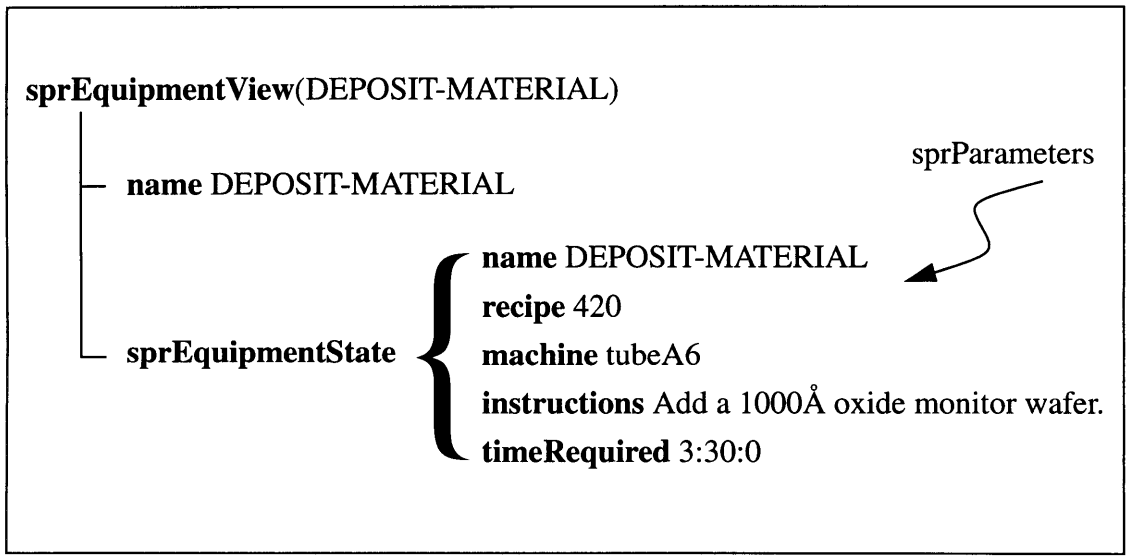


Once a user has somehow loaded a process flow into his editor, he will probably wish to examine or modify parameters which describe a fabrication step. The parameters are contained in sprViews, as described above. A user selects the "view" option of a step

and then chooses the type of view (sprEnvironmentView, sprEquipmentView, or sprEffectsView), he wishes to retrieve from the process repository. Again, this information is sent to the server's CGI for handling. A Tcl script accepts this request and packages the appropriate sprView object for transmission. In extracting PFR information from the database to build an sprView, we utilize only a few PFR object attributes. The sprEquipmentView incorporates the PFR process object attributes machine, timerequired, settings, and instructions. The sprEnvironmentView extracts information from the treatment attribute, and the sprEffectsView is constructed from parsing the changewaferstate attribute.

An illustration of the structure of a packaged sprEquipmentView is given in Figure 4.6. As specified in the SPR standard, the sprEquipmentView is composed of sprEquipmentStates, which are described in turn by sprParameters.

Figure 4.6 sprEquipmentView of DEPOSIT-MATERIAL



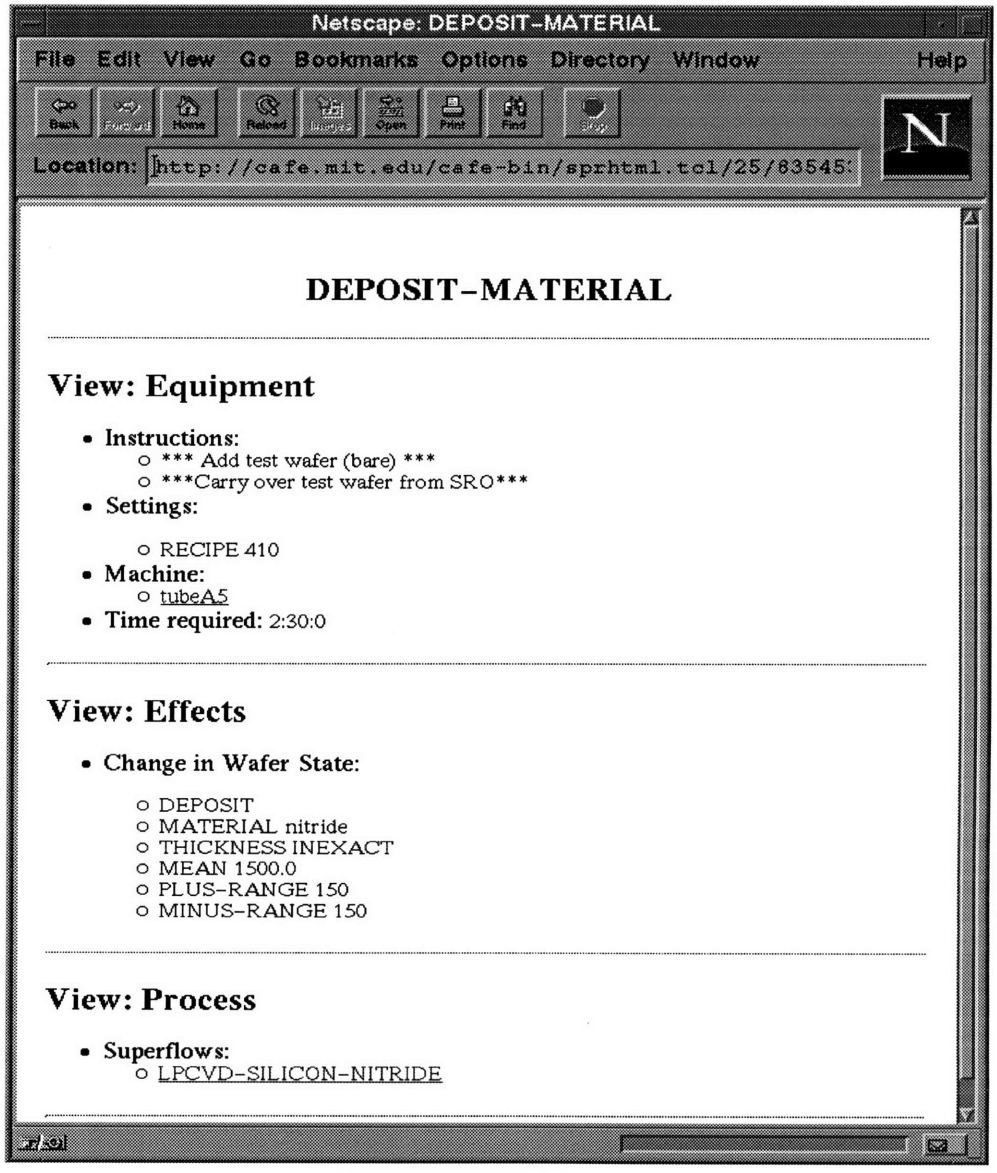
4.2.3 Providing a Process Library

A final development task on the server side lay in enabling the presentation of a sort of library structure of process information. A user may not know the specific names of the process steps she desires to obtain from a remote process repository. A simple mechanism for graphically presenting a process library to the applet user is to format a library of steps in a similar manner to a regular process flow. Instead of representing a tree of distinct and sequential processing steps, this library tree would display specific steps as the "chil-

dren" of general categories like Metallization or Etch (as shown in Figure 4.12).

The library may also be presented as text in a fashion similar to the database traversal system described in Chapter 3. In fact, given our method of extracting PFR data to create SPR views, it was an easy task to modify the process viewing Tcl script of Chapter 3 to display an SPR process. An example of a step in this HTML process catalog is shown in Figure 4.7.

Figure 4.7 A step in the HTML SPR process catalog



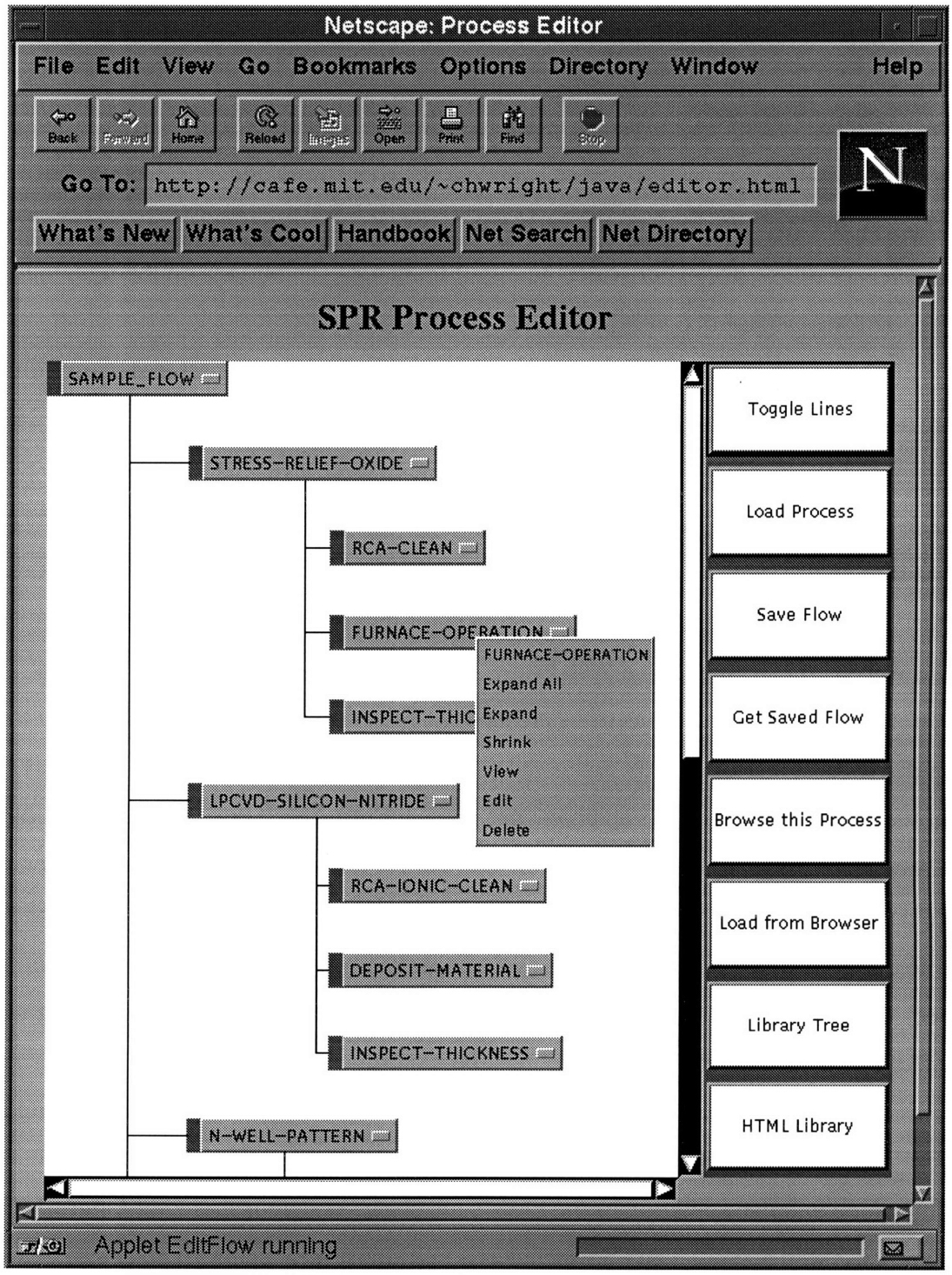
4.2.4 The Java applet

The goal on the client side was to produce a tool for graphically designing and editing process flows. The motivations for achieving this goal with a Java applet have been discussed. The concept of a process design tool implies the ability to import arbitrary process steps or flows, view and structure a process flow, and view and edit the internal parameters of a step. We have developed a Java applet which satisfies these specifications. A screen shot of an editor is provided here, in Figure 4.8. The buttons on the right side of the applet form the user's control panel.

When a user initially opens the process flow design tool, she will find blank editing space and the control panel. The "Load Process" and "Get Saved Flow" buttons initiate mechanisms for loading a flow into the editor. The first retrieves data from the CAFE database, while the second returns a process previously saved to a file on the server. The action directive associated with these buttons causes a dialog box to appear on the screen to query the user for a process name. A request for the named process is then transmitted to the Web server by imitating a Web browser and making a socket connection to the server daemon. The applet thus takes advantage of the normal operation of the Web to deliver a request to a CGI script and receive a response. As described in the previous section, such a response arrives as an object packaged into an ASCII buffer. The Java support of Carney's message and object manipulation restricts the data in object slots to be one of three data types, ASCII, 32 bit integer, or 32 byte floating point. We make use of the first two of these. A process object is recursively unpacked by gathering information from the expected slots (PFR object ID, name, step number, level) and recognizing that a slot with an unknown name and an ASCII value must be a subprocess which needs to be unpacked.

An object is unpacked into a Java vector (dynamic array) of steps representing the process flow. Each sprProcess corresponds to a node in the tree. We maintain information about a node in an instantiation of the FlowNode class. As demonstrated in Figure 4.8, each process step is graphically presented as a pull-down menu; this graphical aspect of a process step is the result of defining FlowNode to be a subclass of the Java Choice class. In addition to defining the applet's presentation of a step, a FlowNode maintains the structure of an SPR object. Detailed information describing the SPR classes that we have implemented can be found in Appendix A.

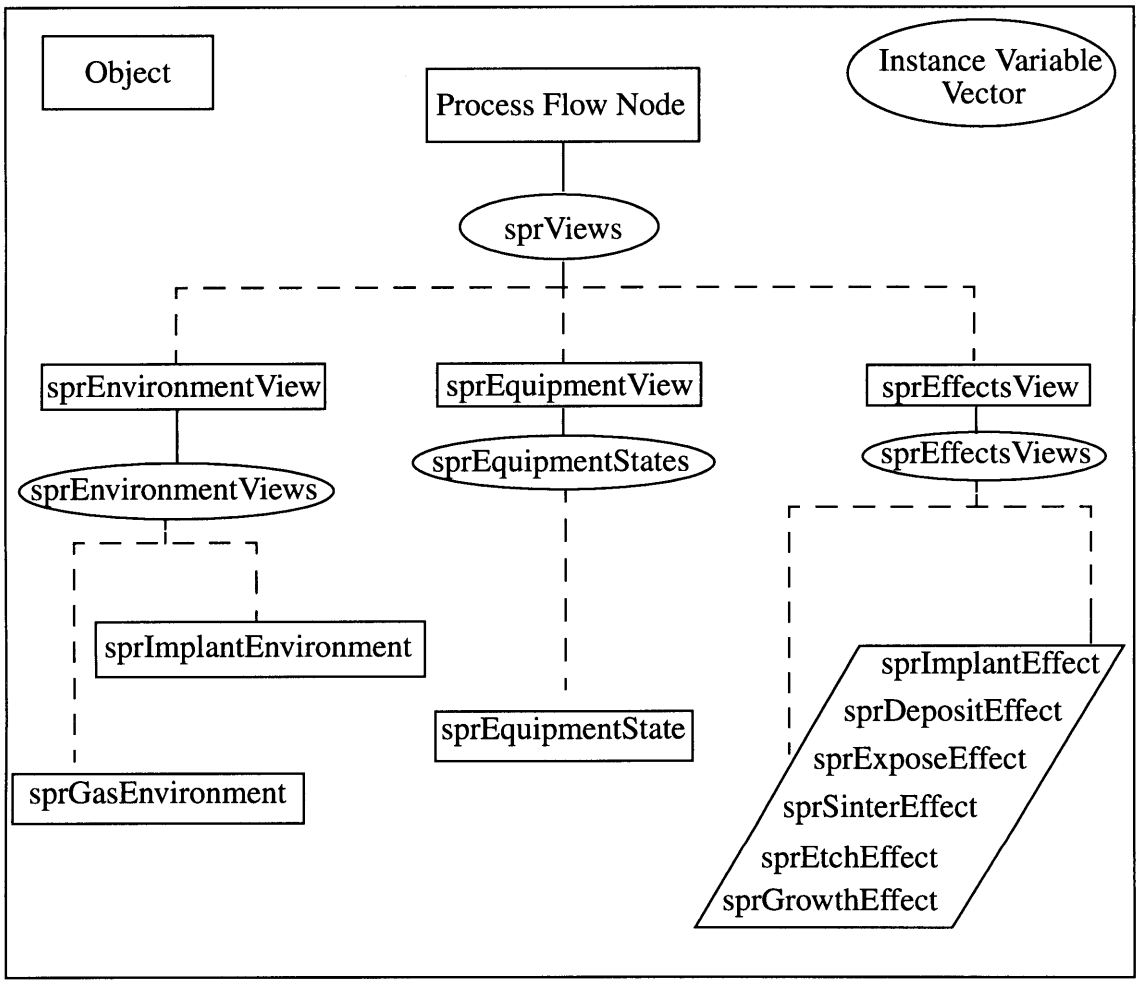
Figure 4.8 Screen shot of the process editor



Since SPR objects are defined by various associated views, it is the view organization that we must preserve. We do not maintain an sprProcessView since the graphical display of the tree always provides this view. Further, it would be simple to derive an

sprProcessView from the vector of nodes associated with a process. Information describing the other views (sprEnvironmentView, sprEffectsView, and sprEquipmentView) is maintained by each FlowNode in an instance variable, sprView, which is a vector of sprViews. The classes corresponding to these views have been implemented consistently with the SPR standard. Figure 4.9 depicts the SPR structure contained within each FlowNode.

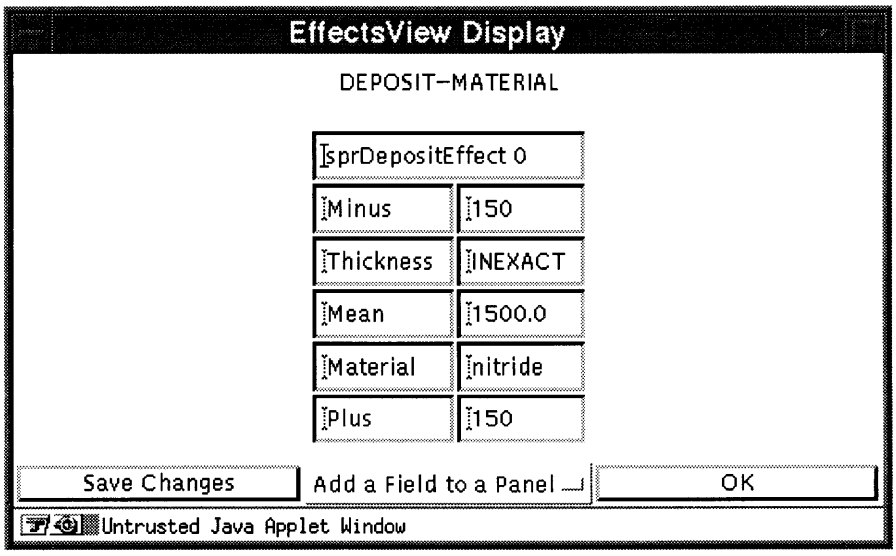
Figure 4.9 SPR data as managed by a process flow node in the applet



Once a user has loaded a flow into the editor, he may reorder or delete steps, and he may edit the internal parameters of a step. This is accomplished by selecting the "Edit" option from the pull-down menu of a step. A dialog box is used to ask the user to select a view to edit. Similarly to the process of fetching a process, the applet sends a request to the Web server. Two parameters are sent to the CGI script; first, the view type requested,

and second, the PFR ID number associated with the step. The server uses this data to retrieve and return an ASCII buffer describing the desired sprView. The applet unpacks the information into an appropriate sprView object and associates it with a FlowNode. To allow the user to edit the view, a dialog box presenting all of the sprParameters as editable TextFields is constructed. A provision for adding additional sprParameters is included in this editing frame, an example of which is shown in Figure 4.10. The parameters in this figure are clearly associated with the PFR, wherein thicknesses can be represented as an "INEXACT" value, a mean plus or minus another value. This information was extracted from the PFR process attributes and forced into self-determined sprParameters (name/value pairs); this method puts the data into the SPR structure but would probably result in a version of SPR too similar to PFR and not adequate for universal usage.

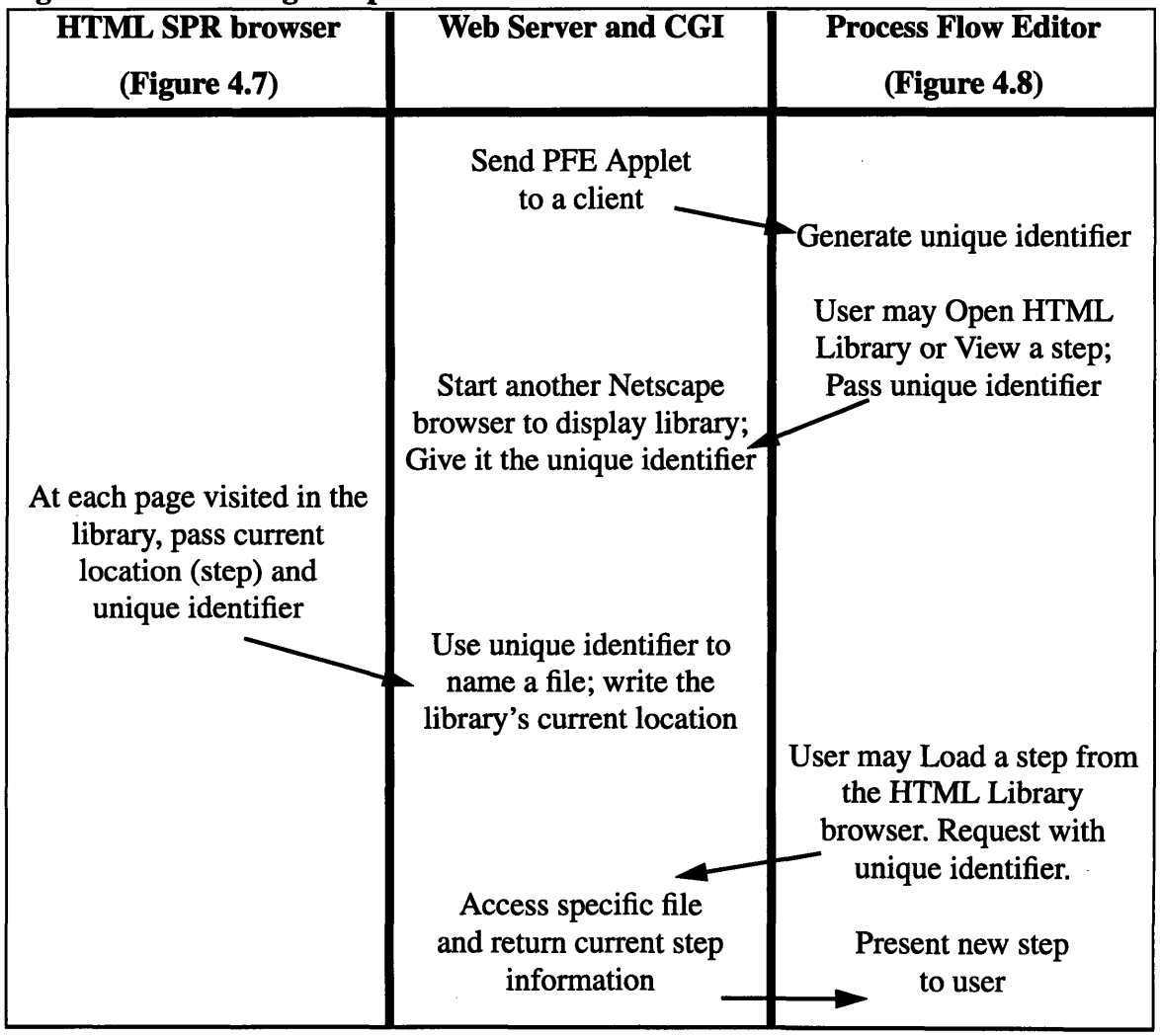
Figure 4.10 Editing an sprEffectsView



In addition to loading a full flow into the editor, a user may wish to load a single step from the repository. The "Load from Browser" button is meant for precisely this purpose. If a user is browsing the HTML repository of steps, as in Figure 4.7, activating this button in the applet results in the retrieval of the step currently in the browser. The support for this action is described by Figure 4.11. Java does not allow applets to acquire information about the state of other Web browsers, so we maintain information describing the user's location in the library by hiding a simple applet in each page of the HTML library.

Whenever a process editor is initiated, a unique identification number is created to refer to that editor. When an HTML library is entered from the editor, that identifier is passed as a parameter to be used with the hidden library applet. At each new page viewed in the library, the hidden applet connects to the Web server and passes the editor identification and the library location to a CGI script which maintains this information in a local file. When a user loads a step, the editor queries the server for the current library step in that file and receives the appropriate sprProcessView. This method obviates the need for developing a daemon on the Web server to handle information transfer. (If Java sockets were usable across arbitrary network links, the two applets could simply communicate this way; however, this is another impassable security restriction imposed by the browser.)

Figure 4.11 Loading a step from an HTML SPR browser



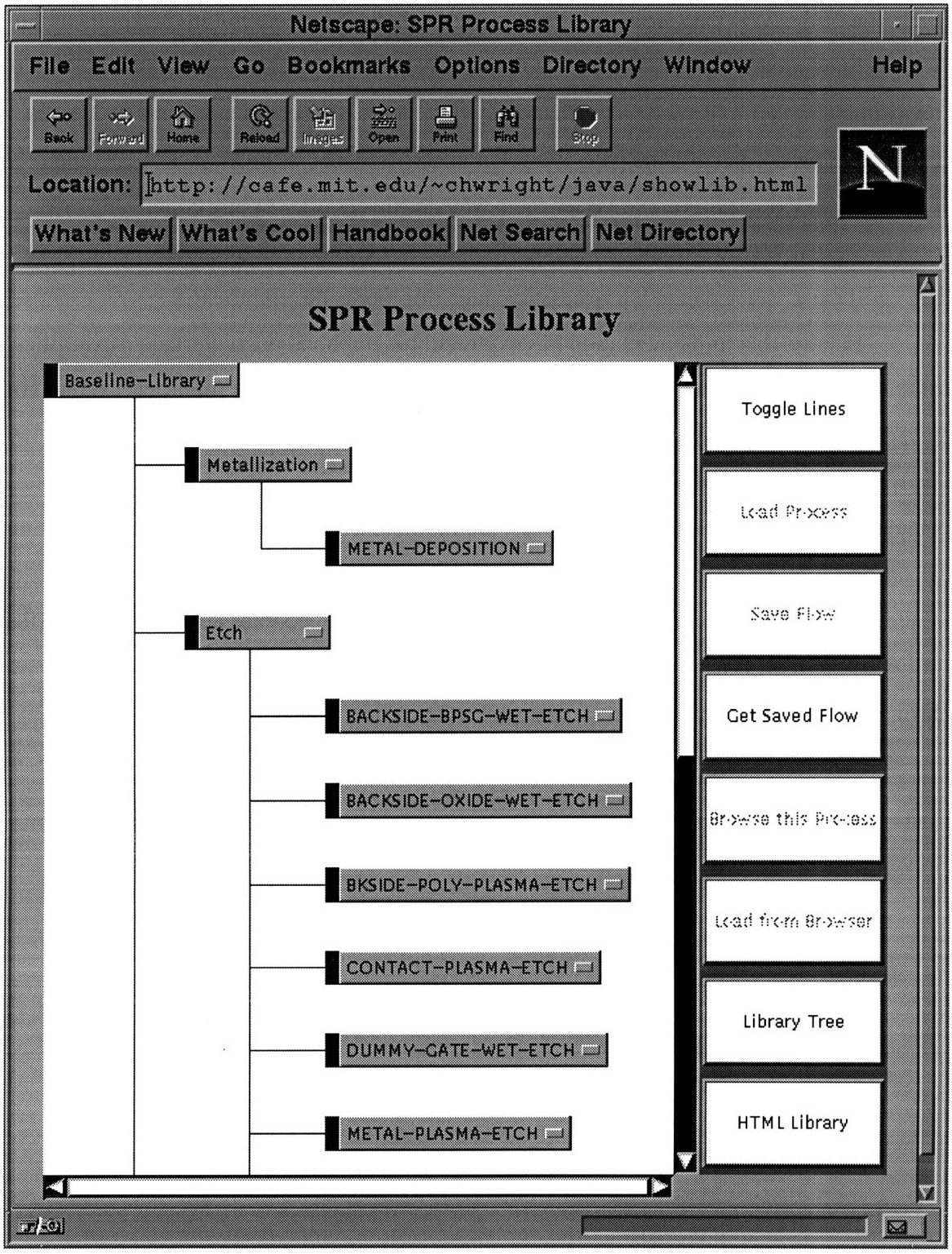
A simple extension to the tree editor makes it possible to graphically view (and edit) the steps within a repository (as opposed to steps within a process). We have made a provision in the editor for displaying a library tree which is packaged by a CGI script on the server just as is a regular sprProcess. This manner of display provides a user with a top-down view of a process repository. For example, the display in Figure 4.12 contains a process "Baseline-Library" which has a child "Etch", which in turn has children representing and describing the etching steps in existence within that repository. For a specified group of users, enabling the process editing capabilities within the library may provide a simple means to update and maintain the "library object" associated with a repository.

Finally, a user will likely wish to save a flow. We have previously discussed security restrictions which prevent us from allowing a user to save a flow to her local machine. We may only write a file to the machine that originally delivered the Java applet. We write a flow by packaging it in the format defined by Carney's messaging system. The format is exactly the same as before, but now we package the sprProcess information and the sprView information together. This ASCII buffer is transferred to a CGI script on the Web server which stores the data in a file named by the user. The file is henceforth available for retrieval by a process editor.

4.3 Conclusion

The process flow editor is functional as described in this chapter. One severe limitation is that a developed flow can only be "saved" to a file on the Web server; we have not attempted to write new flows back to the CAFE database, and the current browser-imposed security restrictions prevent us from enabling a user to save a flow to her local machine. This tool does not write new flows to CAFE largely because of the lack of an exhaustive PFR to SPR (to PFR) conversion. When these conversions are implemented (at MTL as well as other research facilities), we imagine that data from an SPR application such as this flow editor tool will be passed to an intermediary conversion program which will interface to the local data representation, in our case the PFR.

Figure 4.12 A library of SPR processes



In addition to the task of appropriate data conversion, there are several areas where additional work is needed with respect to this tool. First, this editor does not allow a user

to instantiate an entirely new step, but to edit previously existing steps and flows. Again, the issue of the SPR data structures arises; a full SPR implementation in the applet and a well-defined method for building a usable process step are necessary. Of course, it would be a simple matter to provide a mechanism whereby the user could create a new step by entering arbitrary name/value pairs as sprParameters for the various views; we would rely on some validation program to verify the new data (this is true for any edited step or process as well!). We have implemented our tool under the assumption that any "rule checking" will exist as a separate but necessary stage of process development.

Finally, this implementation has explored the graphical user interface capabilities of the Java programming language. As stated earlier, the first official (non-beta) version of Java has been released and was used in this work. The language is yet in its infancy, however, and only its rudimentary graphical widgets are commonly available (there are no standard graphic libraries). This capability was sufficient for our needs, with the exception of actions involving scrolling. Especially as larger process flows are loaded into our applet, scrolling or moving steps (Java Component objects) becomes an impediment to the utility of the tool. It is hoped that fundamental improvements in this area will be made as Java and its support (by browsers) undergo further refinement.

Chapter 5

Conclusion

This thesis has explored the application of Web and supporting technologies to various levels of information sharing in an effort to explore new possibilities for distributed semiconductor research and development.

First, we discussed the use of the Web to create a centralized access point for information relevant to researchers. A user can start at this point and follow hyperlinks to traverse the wealth of distributed information. The well-used Semiconductor Subway demonstrates that simplified accessibility of static documentation does indeed facilitate and speed information propagation through the community.

Next, we described work in the industrial research environment at Intel which produced an automated system to handle organization and dissemination of "miscellaneous" and regularly updated information. The system was designed such that users throughout the research community in PTD could contribute relevant information to the repository. The repository met the goal of maintaining a single, official version of published information at one well-known location: an internal Web server. Such distributed publication of unstructured but crucial information could enhance research in a geographically dispersed community of academic researchers.

Our next focus was to provide coresearchers with information and data relevant to processing at MTL. We developed a system which enabled easy access to the dynamic data in the MTL processing database and included links to other relevant information not

explicitly contained within the database. This system demonstrates the use of the Web to make a cohesive presentation of data from several sources. We discussed in detail the mechanisms for handling the highly structured CAFE data, a different task from presenting the information in the repository described above. Since an academic data representation standard is not in place, we present the data in its native format, the PFR. We acknowledge that increased utility will result from such a system when a standard such as the SPR becomes well-used in the community; however, even this level of dynamic information sharing can be immediately useful to encouraging distributed research efforts.

Finally, we have described an interactive Java applet which implements a process flow development and design tool. We have made an initial implementation of the SPR in Java and attempt to present data in that format, although we extract it from CAFE in the local PFR. If the SPR were thoroughly implemented throughout the research community, the tool could provide a means for a developer to incorporate data from distributed resources in designing semiconductor process flows. In this way, the Java-based tool provides a glimpse of the future, wherein geographically separate researchers could access process information for development and design through a network of distributed site-specific process libraries.

This future relies on the successful implementation of a standard data representation. When that is in place, the geographically dispersed research community will become more cohesive as data is actively and simply shared among process designers, simulators, and fabricators. We have demonstrated that the Web and Java are both technologies which can be effective in realizing this type of information sharing.

References

[Ber94] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, "The World-Wide Web," *Communications of the ACM*, Vol. 37, No. 8, August 1994.

[Bon90] D. S. Boning and M. B. McIlrath, "Semiconductor Process Representation Information Model Overview," Draft - V0.3.18, March, 1994.

[Bon93] D. S. Boning, *A Tcl Interface to GESTALT*, Massachusetts Institute of Technology Internal CIDM Memo Series, No. 93-4, March, 1993.

[Cam95] <http://java.sun.com/tutorial/index.html> "The Java™ Tutorial: Object-Oriented Programming for the Internet" by Mary Campione & Kathy Walrath, Draft V2-V5, October-March 1995.

[Car95] J. C. Carney, "Message Passing Tools for Software Integration," S.M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995.

[Fla96] D. Flanagan, *Java in a Nutshell*, O'Reilly & Associates, Inc., Sebastopol, CA, 1996.

[Gos95] <http://java.sun.com/whitePaper/java-whitepaper-1.html> "The Java™ Language Environment: A White Paper" by James Gosling & Henry McGilton.

[Hey89] M. L. Heytens and R. S. Nikhil, "GESTALT: An Expressive Database Programming System," *ACM SIGMOD Record*, Vol. 18, No. 1, March, 1989.

[Los96] P. Losleben and D. S. Boning, "A New Semiconductor Research Paradigm using Internet Collaboration," *SISPAD '96*, Tokyo, Japan, September 1996.

[McI90] M. B. McIlrath and D. S. Boning, "Integrating Semiconductor Process Design and Manufacture Using a Unified Process Flow Representation," *Proceedings of the 2nd International Conference on CIM*, May 1990.

[McI92] M. B. McIlrath, D. E. Troxel, M. L. Heytons, P. Penfield, Jr., D. S. Boning, and R. Jayavant, "CAFE- The MIT Computer-Aided Fabrication Environment", *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, Vol. 15, No. 2, p. 353, May 1992.

[Ous94] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley, Reading, MA, 1994.

[Pri94] S. Price, "Process Flow Representation to Web Interface," Advanced Undergraduate Project, Dept. EECS, MIT, Dec. 1994.

[Sun95] Sun Microsystems, Inc., "The Java™ Language Specification," Version 1.0 Beta, October 1995.

[Woo93] A. R. Woo, "A Generic Graphical Tree Editor for Wafer Processing," S.M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1993.

Appendix A: Java SPR Implementation

A.1 The Java SPR classes

In developing SPR classes in the Java programming language, we have attempted to preserve, as much as possible, the structure put forth in the SPR standard [Bon90]. However, we have not attempted an exhaustive implementation of the SPR; we have constructed and developed classes sufficient for our demonstration.

As described in Chapter 4, we do not maintain each FlowNode as an sprProcess itself; rather, we have left the construction of the sprProcessView to the future. (This would be a simple task given the Vector of FlowNodes associated with a process.) Each FlowNode has an instance variable, sprViews, a Vector which holds three objects: an sprEnvironmentView, an sprEquipmentView, and an sprEffectsView. As specified by the SPR standard, each of these views is a descendant of sprView. First, we note that since Java does not allow multiple inheritance, we have used Java interfaces for extensions such as this. Second, rather than use a typed array for instance variables like sprViews, we have chosen to use a Java Vector, which behaves like a growable array, but which is not assigned a type. Therefore, an sprViews Vector contains three objects, each of which we expect, but do not know, to be descended from sprView. We use the Java Vector rather than an array for the advantage of ignoring the size and indexing of the array.

The class definition for an sprView is shown here:

```
public interface sprView extends sprExtensibleNamedObject {
        public String getViewType(sprView view);
}
```

This class matches that described in the Base Information Model. First, we note that sprView is defined as an interface, rather than a class. This is consistent with the directive that sprView be an "abstract supertype" of the data-containing views. The interface extends sprExtensibleNamedObject, which is itself an extension of sprNamedObject and sprExtensibleObject. The sprNamedObject assures that each object in the SPR model can have a string identifier associated with it. We did not implement a body of the sprExtensi-

bleObject interface, but included an empty definition to be filled in later as needed. Finally, since every sprView has a type (e.g., Environment, Effects), the sprView interface guarantees that the ViewType is accessible. The sprEffectsView is shown here; it is representative of the instantiable sprView classes (Effects, Environment, Equipment).

```
public class sprEffectsView implements sprView {
        public Vector sprEffects = new Vector();

/* ** The rest implements the inherited interfaces. ** */
        String name;
        static final String ViewType = "Effects";
        public synchronized String getName() {
                return name;
        }
        public synchronized void setName(String s) {
                name = s;
        }
        public String getViewType(sprView v) {
                return ViewType;
        }
}
```

This class implements the methods declared in the interfaces that it derives from. It implements methods for getting and setting the `name` string and for getting the view type. The variable `ViewType` is static and final; it is "Effects" for every sprEffectsView. The instance variable of the sprEffectsView class is a Vector of sprEffects. Again, we use an untyped Vector rather than a typed array.

The SPR standard dictates that sprEffect is an abstract supertype of all specific Effect objects such as sprDepositEffect, sprEtchEffect, etc. Consistently with the standard, the sprEffect assures that each of these effects will have a wafer location associated with it.

```
public class sprEffect extends sprEffectsView {
        String effectlocation;
        public Vector sprEfParam = new Vector();
}
```

It is at this level that we have taken some liberties in implementing the SPR standard; we have considered our specific task of converting PFR to this SPR format. First, given the PFR data format we derive data from, we never capture any `effectlocation` information. We have also specified that an sprEffect (and similarly, an sprEnvironment or an sprEquipment) has a Vector of sprParameters which contain the data.

Each of the instantiable sprEffects, such as sprEtchEffects is defined to contain parameters and information relevant to that Effect specifically. The sprEtchEffect, as defined by the SPR standard, has variables `Material` and `Thickness` (to etch).

```
public class sprEtchEffect extends sprEffect {
        String Material;
        String Thickness;

        public Vector sprEfParam = new Vector();

        public String getMaterial() {
                return Material;
        }
        public void setMaterial(String m) {
                Material = m;
        }
        public String getThickness() {
                return Thickness;
        }
        public void setThickness(String t) {
                Thickness = t;
        }
}
```

A thorough PFR to SPR translation would involve parsing all PFR data associated with a PFR object to determine and fill the appropriate SPR objects (such as sprEtchEffect). Instead, as described in Chapter 4, we have queried CAFE for specific attributes of a PFR object and then used those attributes to fill a simple SPR structure. We perform a simple, inexhaustive mapping of the PFR attribute changewaferstate to sprEffects. We have specifically implemented sprDepositEffect, sprEtchEffect, sprExposeEffect, sprGrowthEffect, sprImplantEffect, and sprSinterEffect. If the PFR attribute does not fit into one of these Effects, we simply use the general sprEffect. Then, instead of parsing the attribute to fill the Effect-specific slots such as "Material," we store the information in sprParameters which are simply name/value pairs.

These simplifications in implementing the SPR have allowed us to present PFR data in an SPR format without undertaking the task of an exhaustive PFR to SPR conversion. We have tried to maintain the standard structure even where we did not utilize it in order to simplify future efforts to realize a thorough and accurate conversion.