MASTER COPY KEEP FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1997	Technical Report

4. TITLE AND SUBTITLE
OmniDesk and OmniFlows: Platform-Independent Executable and User-Reconfigurable Desktops...

5. FUNDING NUMBERS
DAAH04-94-G-0280

6. AUTHOR(S)
H. Lavana, F. Brglez

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
Collaborative Benchmarking Laboratory
Department of Computer Science
North Carolina State University
Box 7550 NCSU
Raleigh, NC 27695-7550

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park,, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 33616.6-EL

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

19980521 113

13. ABSTRACT *(Maximum 200 words)*
Today, web browsers provide a convenient access to the Internet while
(1) increasing the number of useful desktop functions, and,
(2) reducing the platform dependence on the operating system of the host. This paper introduces OmniDesk, implemented as an applet, that creates a user-configurable desktop within the web browser window. User can place any number of objects onto the OmniDesk, ranging from windows that display the contents of a directory or a file on a remote host, to OmniFlow applets that can execute any sequence of user-defined and data-dependent tasks. Identical versions of OmniDesk and a variety of OmniFlow class libraries can be mirrored on several web sites or can be installed locally for faster access and execution.

An OmniFlow is a user-created directed dependency graph of data, program, decision, and OmniFlow nodes. Data and program nodes may reside anywhere on the Internet. The proposed approach has a number of advantages over the current html-form-based execution of CGI programs and applets. Most significantly, the OmniFlow captures, hierarchically, any number of user-defined and data-dependent task sequences, including ones that have cycles -- a feature that would be impractical to implement with current html-form-based approaches. The data and program node configurations consist of one-time-only form entries which can be used and re-used in any number of OmniFlows.

14. SUBJECT TERMS	15. NUMBER IF PAGES
Internet, desktops, frameworks, web browsers, html-forms, CGI scripts, applets, Security	7
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CBL (Collaborative Benchmarking Laboratory)
Department of Computer Science
Campus Box 7550
North Carolina State University
Raleigh, NC 27695

OmniDesk and OmniFlows:

A Platform-Independent Executable and User-Reconfigurable Desktops and Workflows on the Internet

Hemang Lavana Franc Brglez

Technical Report 1997-TR@CBL-05-Lavana
October 1997

If you choose to cite this report, please add the following entry to your bibliography database:

```
@techreport{
1997-TR@CBL-05-Lavana,
author = "H. Lavana and F. Brglez",
title  = "{OmniDesk and OmniFlows:
A Platform-Independent Executable and User-Reconfigurable
Desktops and Workflows on the Internet}",
institution = "{CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695}",
number = "1997-TR@CBL-05-Lavana",
month  = "Oct",
year   = "1997",
note   = "{Also available at http://www.cbl.ncsu.edu/publications}"
}
```

To contact the Collaborative Benchmarking Laboratory via Internet, you may consider:

WWW :	`http://www.cbl.ncsu.edu/`
Anonymous FTP :	`ftp://ftp.cbl.ncsu.edu`
For an auto-reply:	`benchmarks@cbl.ncsu.edu`

To deposit a file at CBL:	`ftp cbl.ncsu.edu`
	`cd /pub/Incoming`
	`put new_benchmark.tar.Z`

Technical Report 1997-TR@CBL-05-Lavana (October 1997)

OmniDesk and OmniFlows: A Platform-Independent Executable and User-Reconfigurable Desktop and Workflows on the Internet

Hemang Lavana Franc Brglez
CBL (Collaborative Benchmarking Lab), Dept. of Comp. Science, Box 7550, NC State U., Raleigh, NC 27695, USA
http://www.cbl.ncsu.edu/

Abstract – *Today, web browsers provide a convenient access to the Internet while (1) increasing the number of useful desktop functions, and, (2) reducing the platform dependence on the operating system of the host. This paper introduces* OmniDesk, *implemented as an applet, that creates a user-configurable desktop within the web browser window. User can place any number of objects onto the OmniDesk, ranging from windows that display the contents of a directory or a file on a remote host, to* OmniFlow *applets that can execute any sequence of user-defined and data-dependent tasks. Identical versions of OmniDesk and a variety of OmniFlow class libraries can be mirrored on several web sites or can be installed locally for faster access and execution.*

An OmniFlow is a user-created directed dependency graph of data, program, decision, and OmniFlow nodes. Data and program nodes may reside anywhere on the Internet. The proposed approach has a number of advantages over the current html-form-based execution of CGI programs and applets. Most significantly, the OmniFlow captures, hierarchically, any number of user-defined and data-dependent task sequences, including ones that have cycles – a feature that would be impractical to implement with current html-form-based approaches. The data and program node configurations consist of one-time-only form entries which can be used and re-used in any number of OmniFlows.

The initial experiments demonstrate the feasibility and advantages of the proposed approach. These include user-configurable OmniFlows of distributed data and distributed university/commercial software, each executable within the web-browser's OmniDesk window.

Keywords: Internet, desktops, frameworks, web browsers, html-forms, CGI scripts, applets, security.

I. Introduction

The design of electronic systems increasingly relies on teams, software, libraries, and technologies that are distributed on a global scale. Given the proliferation of distributed data, programs, and OS-dependent platforms such as UNIX, Windows and MacOS, what users want foremost is *a platform-independent and consistent interface, at least at the higher levels, to interact with distributed tools, data, and users.* The research in this paper explores and expands the Internet to create a platform-independent user-reconfigurable and scalable EDA environment that can support the design needs of distributed teams on a global scale.

Authors from NC State U. were supported by contracts from the Semiconductor Research Corporation (94-DJ-553), SEMATECH (94-DJ-800), and DARPA/ARO (P-3316-EL/DAAH04-94-G-2080).

Web browsers provide a convenient access to the Internet while (1) increasing the number of useful desktop functions, and (2) reducing the platform dependence on the operating system of the host. The *OmniDesk*, as proposed in this paper, creates a user-configurable desktop within the web browser window. User can place any number of objects onto the OmniDesk, ranging from windows that display the contents of a directory or a file on a remote host, to *OmniFlow* applets that can execute any sequence of user-defined and data-dependent tasks. Both the OmniDesk and any OmniFlow are executed as applets in Tcl/Tk [1] and require the user to install a Tcl plugin [2] in their browser. Implementing either or both as applets in Java [3] is feasible but not necessary.

Formalized descriptions of workflow modeling concepts, architecture, and implementation are still evolving, e.g. [4]. Many are application-specific rather than generic. Our approach expands on the attributes of the design control language `decol` [5] devised to deal with a *large* number of *data files* and *programs* and applied to integration of many tools into a comprehensive VLSI design system [6]. Some aspects relate to the work in the area of design methodologies and design frameworks such as [7, 8, 9, 10]. However, with the exception of [11, 12], the earlier research predates the challenges and opportunities of the Internet.

The paper is organized into the following sections:

(2) motivation;
(3) user view of the OmniDesk environment;
(4) user view of the OmniFlow environment;
(5) highlights of OmniFlow scheduling and execution;
(6) security as it relates to OmniDesk and OmniFlow;
(7) summary of current applications; and
(8) conclusions.

II. Motivation

The most frequent use for web-browsers today is to access data on the Internet. The use of html-form-based tools on the web, whether executed as CGI programs on remote hosts or downloaded as applets to execute on the local host, is just emerging. To motivate the introduction of the proposed approach, we contrast the use of html-form-based tools with the more general context of OmniDesk and OmniFlows.

HTML-form execution. We can summarize a typical scope of their use as follows:

- user specifies an URL of the html-based entry form for a specific program (this access may be controlled with a password);
- once the form is available, user may be requested to enter:
  1. names of input data file(s) (using the 'browse' button);
  2. any required/optional parameter values;
  3. a click on the 'submit & execute' button (typically, the program notifies the user as to the location of all relevant output data files);
- once the output data files are available, user downloads them for examination and for potential use as input

data files to any other program in a specific workflow sequence.

Advantages include:

- simplicity of implementation (for the web-page designer);
- convenience for one-time use (for the web-page user).

Drawbacks include:

- user must re-enter the entire entry form to re-execute the same program;
- user needs to download and examine output data files of the web-based program before submitting them to another web-based program;
- user may get overwhelmed when executing a number of complex sequences, especially if they include cycles.

An alternative to HTML-form execution. The various phases of the design of complex electronic systems involve interaction of heterogeneous data (in prolific formats) with a number of complex and specialized tools in several configurations. Potentially, html-form-based applications can fill the need for many point tools. However, there is no single configuration of point tools that can support a complex design process nor are all required tools available from a single vendor and a single web site. Consider the scenario such as illustrated in Figure 1(a). Here, we consider the role of three point tools on the web to support a typical segment of the design process:

1. a `partitioner@URL4` extracting `init-partition@URL5` from the circuit description `circuit@URL2`;
2. an `optimizer@URL8` attempting to reduce the size of the initial circuit partition, returning `opt-partition@URL9`;
3. a `place&route@URL10` tool that *conditionally* reads the optimized circuit partition and generates a circuit layout implementation `layout@URL11`.

The dependency graph of data, tools, and decisions in Figure 1(a) can be executed, step-by-step, using the html-form-based approach. However, the process is tedious, error-prone, and hard-to-coordinate with distributed designers, especially in view of the decisions such as `re-partition?` that may lead to a new iteration. We argue that distributed users be given the flexibility to configure dependency graphs such as shown in Figure 1(a) into executable workflows – *OmniFlows* which are rendered platform-independent as objects within the web-browser window of the *OmniDesk*. We describe both concepts in this paper in more details later. First, we illustrate the context for their use in Figure 1(b). Here, we depict two users, `user1` and `user2`, engaged in very different tasks. The main objectives of `user1` are:

1. to download the applet `OmniDesk@cbl`;
2. to *configure* a specific OmniFlow that uses programs such as `program2@mit` and data such as `data1@ucb`;
3. to execute the OmniFlow and archive output data as `ResultsReport1@anywhere`.

On the other hand, `user2` appears to have a local copy of OmniDesk already. Her objectives are:

1. to browse `OmniFlow-lib@mpi` and download a specific OmniFlow;
2. to execute the specific OmniFlow and archive output data as `ResultsReport2@anywhere`.

Both scenarios above are extreme cases. In general, user may start with an existing OmniFlow and modify it as appropriate.

Scope of OmniDesk/OmniFlows. A desktop that supports *drag and drop* of a file onto the printer icon is a simple example of a universally understood paradigm. Such a simple task sequence can be easily remembered. However, typical design tasks consist of longer sequences, involving data, programs and users anywhere on the Internet. OmniDesk, implemented as an applet, creates a user-configurable desktop within the web browser window. User can place any

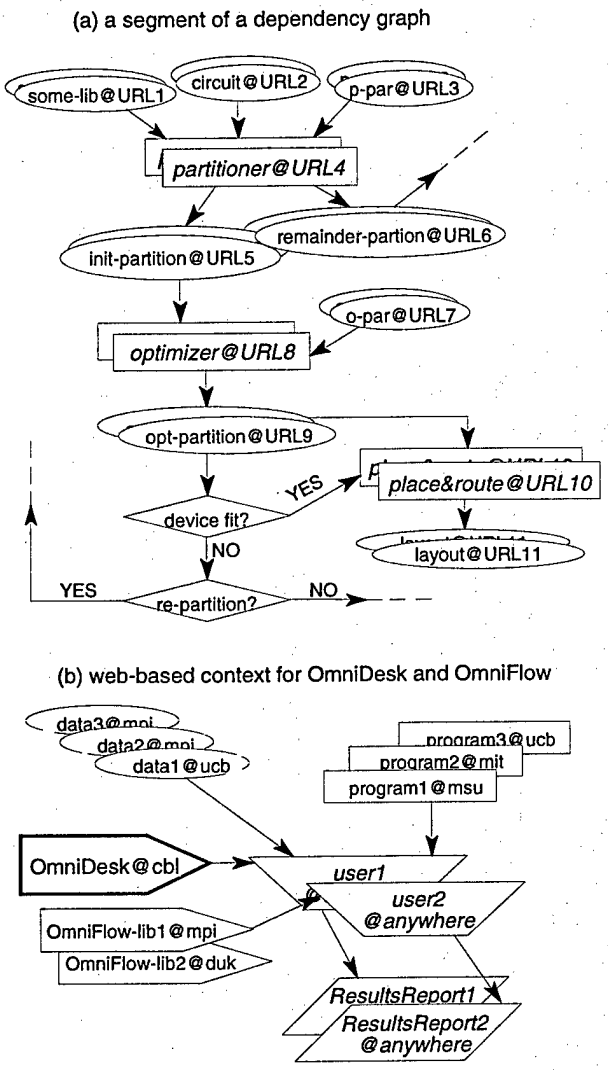

Fig. 1. Dependency graph, OmniFlow and OmniDesk.

number of objects onto the OmniDesk, ranging from windows that display the contents of a directory or a file on a remote host, to OmniFlow applets that can execute any sequence of user-defined and data-dependent tasks. Identical versions of OmniDesk and a variety of OmniFlow class libraries can be mirrored on several web sites or can be installed locally for faster access and execution. The premise of a user-configurable desktop, where partners can collaborate at any time, from anywhere, with anyone, is clearly within reach with this technology.

We contrast the first-time set-up of an OmniFlow with the html-form-based tool execution discussed earlier:

- user specifies an URL of the entry form for a specific program (this access may be controlled with a password);
- once the form is available, user may be requested to enter:
  1. URLs of existing input data file templates or create new ones;
  2. URLs of existing output data file templates or create new ones;
  3. any required/optional parameter values;
  4. a click on the 'submit' button to SAVE the form's content as a specific program template;
- once the templates for all program and data nodes are completed, user creates and executes the OmniFlow

within the OmniDesk window. In general, OmniFlow executes all programs in the workflow specification either sequentially or in parallel.

Advantages include:

- convenient for repetitive applications, since each form is entered only once and saved as a configuration file;
- URL for the entry form is never accessed during the execution, reducing the network traffic;
- output data files of any web-based program are transferred automatically as input data to another web-based program;
- OmniFlow captures, hierarchically, any number of user-defined and data-dependent task sequences, including ones that have cycles, a feature that would be impractical to follow-through with the current web-form-based approaches;
- libraries of OmniFlows, each executable within the OmniDesk, can be created and maintained at any web-site.

Drawbacks include:

- relatively high overhead to set-up the OmniFlow for one-time use.

Combined Approach. CGI programs and applets available on the web for html-form-based execution are a valuable resource. The proposed OmniDesk/OmniFlow implementation also supports *the use of html-forms* to automate the one-time-only setup of data and program node configuration templates. These templates in Tcl[1] can then be used and re-used in any number of OmniFlows.

III. OmniDesk – User View

OmniDesk may be invoked by downloading the Tcl applet from a web-server using the web-browser such as Netscape or Internet Explorer. Upon invocation, the OmniDesk appears as a scrollable window embedded inside the web-browser and occupies the entire size of the web-browser, as shown in Figure 2. Initially, it contains only two windows - the OmniDesk Manager and the OmniBrowser.

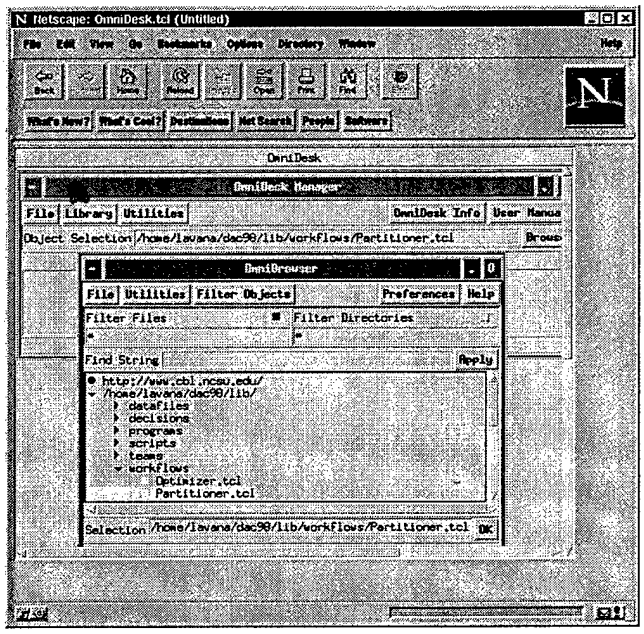

Fig. 2. OmniDesk and OmniBrowser within a web-browser.

The OmniBrowser allows the user to traverse and browse the local file system as well as the various web-sites available on the Internet. On traversing the local file system, the directories and the files are listed as shown in Figure 2. On the other hand, when a URL corresponding to a specific web-site is visited, the contents of the specified URL is parsed for hyperlinks and only the hyperlinks are displayed in the OmniBrowser, in a fashion similar to the directory/file listing of the local file system. A user may then select and invoke any number of pre-configured OmniFlows, which may be located either on the local file system or anywhere on the Internet.

OmniBrowser is hidden from the user's screen as soon as the OmniFlow is invoked. However, to invoke other OmniFlows, a user may retrieve the hidden OmniBrowser window by clicking on the Browse button of the OmniDesk Manager window.

IV. OmniFlow – User View

Any OmniFlow, such as shown in Figure 3, is an executable graph-based representation of a user-specified, data-dependent task sequences. It is represented as a directed dependency graph of data, program, decision, and OmniFlow nodes. The OmniFlow node itself may consist of data, program, decision, and other OmniFlow nodes. Data and program nodes, represented by ovals and rectangles respectively, may reside anywhere on the Internet. Data-to-program edges represent activities such as downloading the data files from URL host to local host and uploading the data files to a URL host where the program node is executed. During execution, each edge is blinking during the file download and all input edges to a program node blink simultaneously during data upload. Program-to-data edges link the executable program node to the output data nodes. Both reside on the same host. The program node can be an applet, downloaded from URL host to local host, or a program residing on the local host. Program nodes can be scheduled to execute serially or in parallel, all are blinking during their execution. Hierarchical nodes, such as Optimizer in Figure 3, expand into OmniFlows of their own as shown in Figure 4.

We next describe the steps involved in creating program and data node configurations to execute a CGI program and contrast it with the existing html-form based invocation of a program

Program Node Configuration. A program node configuration template is either fully created by the user, or it may be generated automatically from the corresponding html-form.

Figure 5 shows a html-form based entry for a CGI program that partitions a given netlist into several smaller partitions. It expects the user to specify two files residing on her local host - an input design file which needs to be partitioned and a device library file. The input design file may be one of several formats acceptable to the CGI program such as 'blif', 'kiss', etc. In addition, a user is expected to supply several other parameters for partitioning the netlist. After entering the html-form, a user typically submits the data for execution by clicking on the `Execute` button. This invokes the corresponding CGI program and upon completion of its execution, it reports the URLs of the generated output data.

We contrast this method with our proposed mechanism. Figures 6 and 7 show an equivalent representation of the html-form based entry as program node configuration and data node configuration templates. To create a program template, such as in Figure 6, a user specifies the URL for the html-form. The Program Manager then fetches the specified URL and automatically generates a graphic representation consisting of several nodes, depending on the contents of the html-form. Specifically, here four nodes are created: `Partitioner`, `New Netlist`, `Device Library`, and `Initial Parameters`.

The `Partitioner` node corresponds to the CGI program

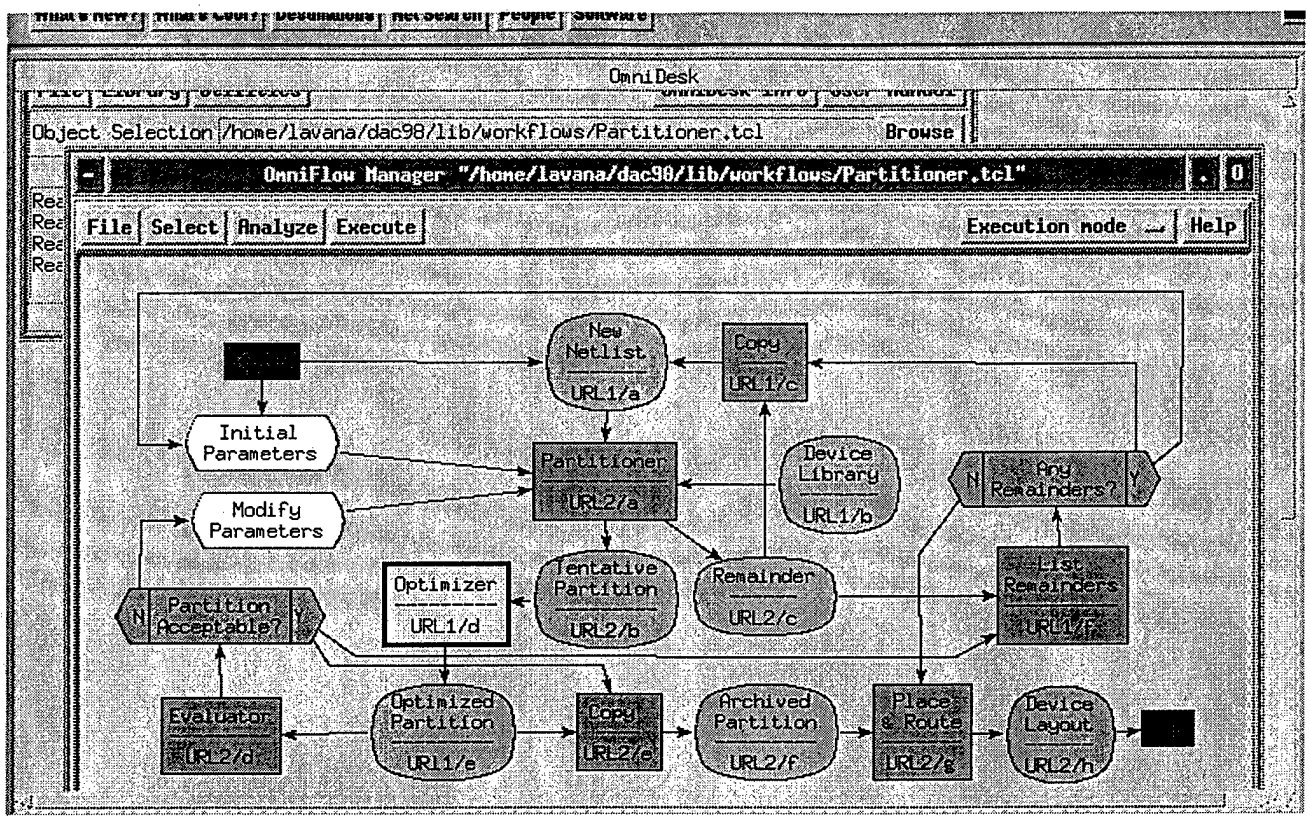

Fig. 3. OmniFlow depicting the partitioning, logic optimization, placement and routing.

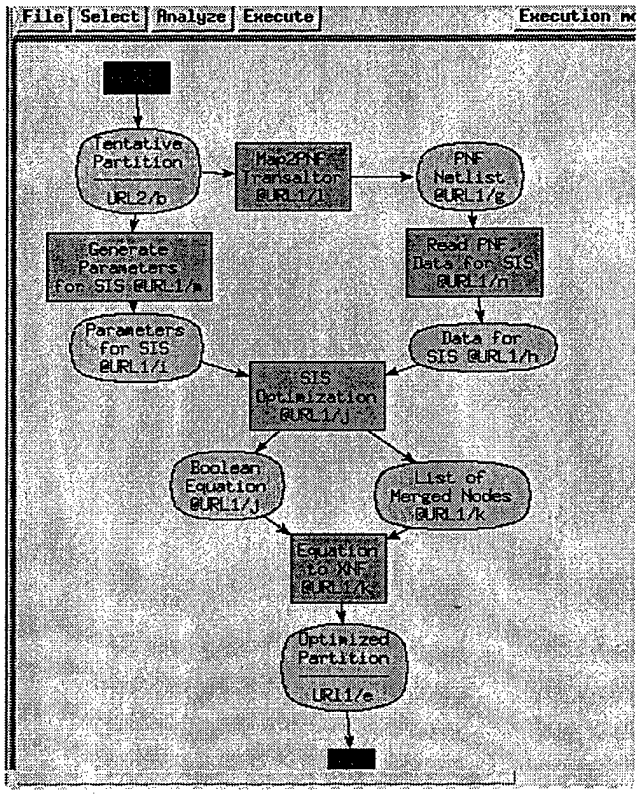

Fig. 4. Expanded view of a hierarchical OmniFlow node.

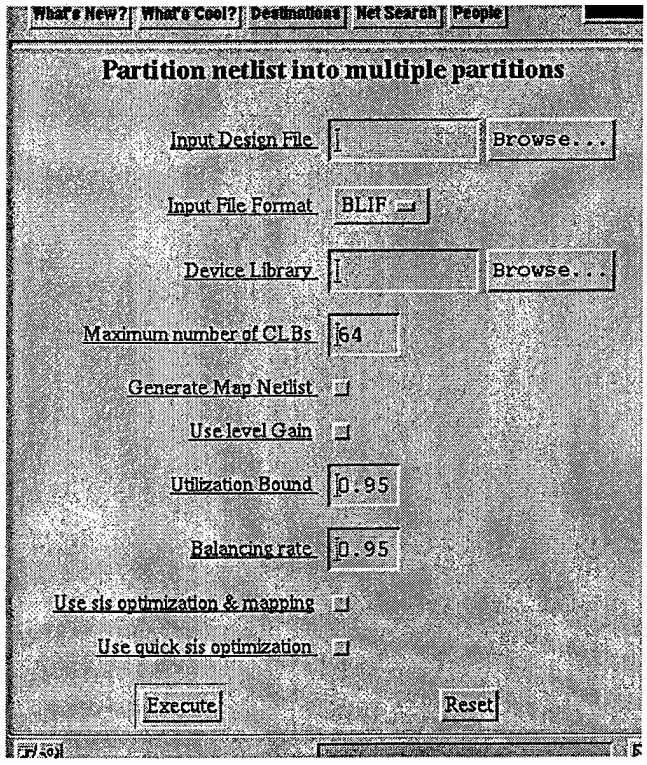

Fig. 5. HTML-form for executing partitioner.

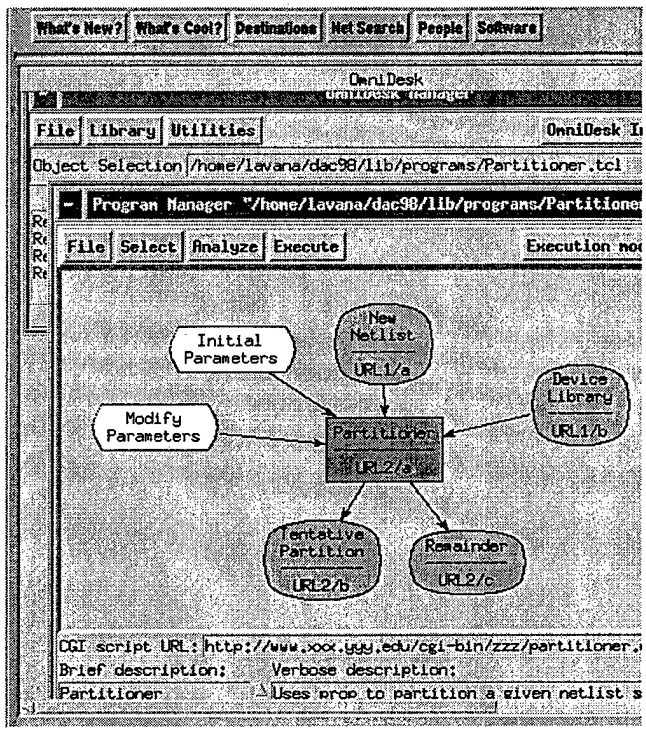

Fig. 6. Program node configuration.

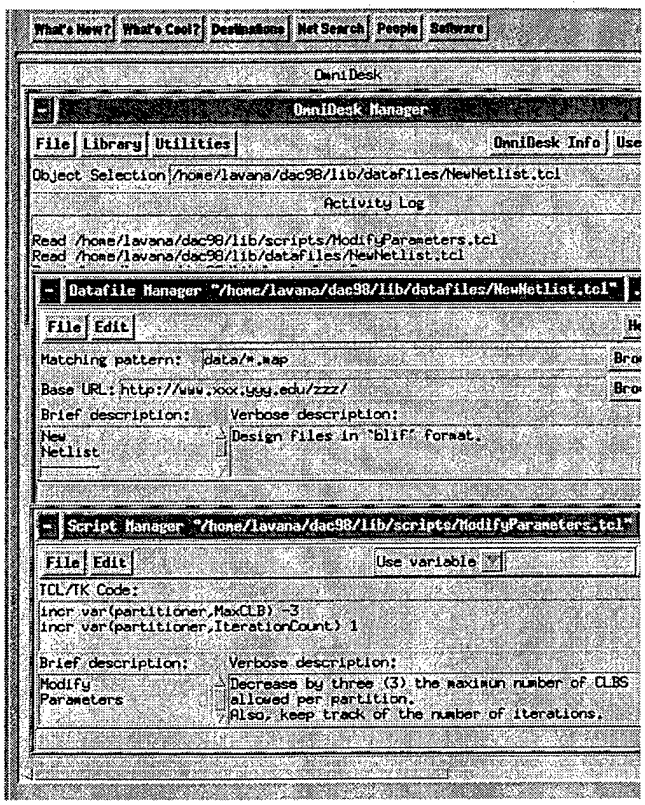

Fig. 7. Data and script node configuration.

located at URL2/a. The `Input Design File` and the `Device Library` are represented as data nodes `New Netlist` and `Device Library` in this template, while all the remaining parameters are specified as script node `Initial Parameters`. Figure 7 shows a data node configuration template for `New Netlist`. It essentially consists of a base URL and a matching pattern to represent a class of data. Thus, in this template, the data is not restricted to reside on the users local host, but can reside on any of the web-sites on the Internet. During execution, one or more data files, based on matching pattern specification in the template, are downloaded and submitted for execution of CGI program. Similarly, data node configuration is specified for the `Device Library`. For remaining parameters, the script node comes up with a window containing all the parameter entries The user is required to specify the default values for these parameters.

The contents of the html-form, however, provides no information as to the output results generated. Therefore, the user is expected to execute the CGI program once through the html-form, determine the URLs of the output data generated and add the corresponding output data nodes in the program template. Depending on the results, the CGI program may be required to be re-executed with the same data but different partitioning parameters. Figure 7 shows a window for specifying a script node template `Modify Parameters` that decrease the maximum number of CLBs and at the same time tracks the number of times the program is invoked with the same data.

Other CGI programs on the web can be similarly accessed by configuring a program template for each one of them. These pre-configured program and data templates can be interconnected with directed edges to construct an OmniFlow and thus specify their sequence of invocation in relation to one another. The next section describes the formalism necessary to schedule and execute an OmniFlow.

V. OmniFlow Scheduling and Execution

A user-created OmniFlow consists of several nodes and edges. Any node, dependent on data generated by execution of other nodes, cannot execute until all its earlier nodes have executed. On the other hand, some nodes can start execution concurrently with any other node, if the two are not interdependent. In addition, a program node may have to be executed several times in a loop before a condition is satisfied.

To schedule OmniFlow for execution, we modified the formal Petri Net based approach in [12] as follows:

(1) Program node templates in OmniFlows are transformed into transitions in a Petri Net, whereas data node templates in OmniFlows are transformed into places in a Petri Net.

(2) All cycles existing in OmniFlows are identified and broken down to form conditional loops.

(3) A firing schedule is generated based on the transformed Petri Net.

(4) The firing schedule is applied to execute the OmniFlow.

Figure 8 shows a feasible firing schedule for the OmniFlow shown in Figure 3.

Security and OmniDesk/OmniFlow

All applets that are downloaded from a web-site and executed inside a window in the web-browser are considered to be unsafe and hence untrusted. However, a safe-Tcl applet has a very limited functionality and can hardly do anything useful. Therefore, the Tcl-plugin supports multiple security policies so that a Tcl applet can perform a variety of useful functions in a safe manner [13].

The standard Tcl-plugin supports several security policies:

Safesock: This allows the downloaded Tcl applets to open network connections to pre-configured list of hosts.

OmniFlow nodes										
Begin	*									
Initialize Parameters		*								
Modify Parameters							*			
Partition Acceptable?						*				
Any Remainder								*		
End										*
Nodes on URL1										
Optimizer					*					
Copy							*			
List Remainder							*			
Nodes on URL2										
Partitioner				*						
Evaluator						*				
Copy									*	
Place and Route									*	
Level of Petri net transition	0	1	2	3	4	5	6	7	8	9

Fig. 8. Firing schedule of an executable Petri net.

Tempfile: This policy lets Tcl applets to store data persistently on the hosting system.

Browser: This policy provides access to web-browser functions such as html display, URL get and post operations, and JavaScript. It also incorporates *Tempfile* policy.

Trusted: All the functionality of the Tcl/Tk is restored, the Tcl applet being assumed to be fully trusted.

These policies are mutually exclusive and a single Tcl applet cannot access more than one policy at the same time. Currently, OmniDesk uses the *Trusted* policy for its execution, since it is designed to operate on data and programs that reside not only on local host but anywhere on the Internet. However, it is possible to use a different policy than *Trusted* with OmniDesk by trading off some of its functionality. For example, one can easily use only the *Browser* policy if one wants to prevent the OmniDesk applet to access the data residing on the local host. This limits the OmniDesk to execution mode only, editing an OmniFlow is not feasible since it cannot be saved.

Applications and Experiments

The applications and experiments, summarized in this section, are the initial part of the OmniDesk environment performance and functionality evaluation. Each of these experiments relies on *interactive* user inputs and is repeated two or three times to determine variations in measured performance. Specifics about the testbed configurations, test cases considered, and tabulated results follow.

Applications. The OmniFlow example in Figure 3 illustrates its potential to customize a *design flow* of distributed tools. It represents one of the Internet-based desktop environments that will be put to test as an integral part of a national level collaborative and distributed design project involving teams at 8 sites. A joint demo of the project, including the desktop described in this paper, is being planned for a demo in the University Booth during DAC'98.

Another set of applications that provides the motivation for this work is related to benchmarking of heuristic algorithms in EDA. The web browser-compatible OmniDesk/OmniFlow environment can facilitate access to benchmarking data archives, can support execution of collaborative peer-reviewed benchmarking experiments, and can provide unbiased evaluation, report generation, and archival of any newly posted results of benchmarking experiments. An example of an OmniFlow to support comparative benchmarking of technology mappers with two participating sites is shown in Figure 9.

Here, both mappers retrieve, from the the archival site, netlist benchmarks to be mapped. Each site is using a distinctive library. The question that arises relates to the performance of each mapper. This can be processed independently

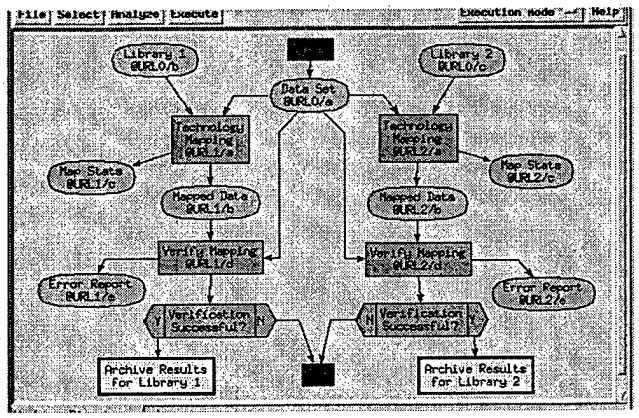

Fig. 9. An OmniFlow to benchmark two technology mappers.

at the archival site, by expanding the hierarchical node such `Archive Results for Library 1` and evaluating submitted data, generating a set of reports, and posting them for web-based review by the peers. We expect to engage distributed participants to conduct experiments such the one described here in the University Booth during DAC'98.

Testbed Configurations and Test Cases. In order to approximate typical instances of a distributed multi-site OmniDesk environment, we have created: *(1) local environment* by installing the Tcl-plugin on our host[1]; *(2) cross-state environment* by installing the Tcl-plugin on a host[2] at Duke University in Durham, NC; and *(3) cross-country environment* by installing the Tcl-plugin on a host[3] at the University of California in Berkeley, CA.

We have carefully selected six test cases of typical short duration OmniDesk sessions, with useful attributes that demonstrate typical user-invoked tasks. The brief description that follows includes the reports of *real_time*, *user_time* and *system_time* as produced by the Unix utility `time`. The 'real_time' corresponds to the 'stopwatch_time' that could have been obtained by the user monitoring the task. The 'user_time' is the time required by the CPU to complete the task. The 'system_time' is the CPU time required by the system on behalf of the task. A brief description of all test cases follows.

(1) Editing-1: Using *OmniDesk*, we *open, and edit*, a simple 4-node, 3-arc OmniFlow by selecting, opening, and closing a single data file node-configuration window.

(2) Editing-2: Using *OmniDesk*, we *open, and edit*, the same 4-node, 3-arc OmniFlow by selecting, opening, and closing a single data file node-configuration window and a single program node-configuration window.

(3) Editing-3: Using *OmniDesk*, we *open, and edit*, the 17 node, 22 arc OmniFlow by selecting, opening, and closing 3 data files and a single program node-configuration windows.

(4) Browsing-1: Using *OmniBrowser*, we *traverse* a directory structure, located on the client's local file system, across 3-levels, with up to 141 items in each directory. The directory structures of all the three clients were made exactly the same for uniform comparison.

(5) Browsing-2: Using *OmniBrowser*, we *select, open, and scroll*, from start to end, the same copy of a text file of about 1000 pages (2.2Mb), located on each client.

(6) Execution-1: Using *OmniDesk*, we *open, and execute*, the hierarchical OmniFlow in Figure 3. As shown, the Om-

[1]SUN SPARC 20 (chip=60MHz memory=64Mb swap=732Mb)
[2]SUN SPARC Ultra 1 (chip=167MHz memory=256Mb swap=288Mb)
[3]SUN SPARC 20 (chip=60MHz memory=96Mb swap=365Mb)

niFlow has 20 nodes and 25 arcs; during execution, the node labeled as optimizer expands into a sub-OmniFlow with 14 nodes and 15 arcs.

For all the six test cases, the OmniDesk applet is downloaded from a web-site on our server.

Evaluation Method. From each of the three hosts, each test case is executed 2-3 times, with an interval of at least 30 seconds between each execution. A log file, generated by time (real_time, user_time, system_time) command, archives timing data for each experiment. Similarly, a log file, generated by sar (system activity report) command, archives the load on each of the three hosts during the execution of these experiments. The log file generated by sar provided the information whether or not both the load on the server and the network was sufficiently stable to accept the 'real_time' and 'user_time' results for tabulation.

TABLE I

SUMMARY OF EXPERIMENTS PERFORMED ON THE INTERNET.

Operation	OUR-host		Duke-host		UCB-host	
Real Time[a]	Min	Max	Min	Max	Min	Max
CPU Time[b]	Usr	Sys	Usr	Sys	Usr	Sys
Editing-1	122.4	125.2	118.7	119.9	127.8	128.8
	4.0	1.0	2.2	0.7	3.9	1.2
Editing-2	157.8	168.3	151.9	153.6	162.5	162.9
	5.2	1.0	5.5	1.3	5.2	1.4
Editing-3	248.1	258.4	232.8	236.2	246.9	247.8
	14.3	1.2	8.4	1.2	14.3	1.8
Browsing-1	131.2	134.5	128.8	129.9	140.0	151.1
	6.2	1.2	3.5	0.8	6.3	1.6
Browsing-2	158.6	167.2	155.3	156.7	167.3	170.4
	28.5	2.0	6.5	0.9	28.6	1.9
Execution-1	337.7	357.8	326.4	328.2	353.1	356.1
	20.4	4.4	5.5	1.3	19.8	4.0

[a] Both minimum and maximum values of 'real_time' are reported.
[b] Only average values of 'user_time' and 'system_time' are reported.

Table IV summarizes results of these experiments, performed on the three hosts. Each cell in the table contains four values. The top two entries report the minimum and maximum values of 'real_time' and the bottom two entries report the average values of 'user_time' and 'system_time' for each experiment.

Summary of Results. The data presented in Table IV allows us to evaluate the performance of web-based OmniDesk environments.

1. The 'real_time' for the three hosts varies, depending on the distance between the OmniDesk web-site and the host to which it is downloaded and the characteristics of the host. Specifically, Duke host consistently reported least execution times, followed by our host and UCB host. This is attributed to the higher performance host at Duke.
2. The variations in minimum and maximum values of 'real_time' for each experiment are negligible since the experiments were performed during the night. However, the same experiments showed significant variations during the day when the network traffic and the host load is unpredictable.
3. Comparing the 'user_time' and the 'system_time' for each host, we find that the our host requires the most CPU time and the Duke host requires the least CPU time. This follows directly from the different types of processors and the configuration of each host.

Observations. The successful completion of preliminary experiments provides us with assurance that the experiments are consistently reproducible on a variety of client hosts, given that the nominal cpu load is small and that the network is stable. Specifically, we confirmed that

- Repeated *real time* executions of experiments, where user-inputs are carefully and consistently entered, gives 'real_time', 'user_time', and 'system_time' performance that is comparable (within 10%) of one another – provided that the server load and network conditions are favorable.
- The performance of the web-based OmniDesk environment is quite good under nominal network traffic and load on the server. Hence, with sufficient network bandwidth and powerful processors, it is possible to work efficiently and effectively on a design project, via the web-browser, where the tools and data are dispersed across the continent.

CONCLUSIONS

We have proposed and implemented a platform-independent desktop environment (OmniDesk) on the Internet. This environment supports user-configurable OmniFlows of distributed data and distributed university/commercial software, each executable within the web-browser's OmniDesk window.

The initial applications are driven by the motivation to support a distributed university-based design project and a series of distributed benchmarking experiments. The initial experiments demonstrate the feasibility and advantages of the proposed approach.

ACKNOWLEDGMENTS. We appreciate the access to remote servers and tools used in this research: user accounts on two remote servers, facilitated by Dr. Richard Newton at UC Berkeley and Dr. Gershon Kedem at Duke U., SIS and WELD from teams at UC Berkeley, PROP from Roman Kužnar from U. of Ljubljana (Slovenia), and APR from Xilinx Inc.

REFERENCES

[1] The Tcl/Tk Home Page. Published under URL: http://sunscript.sun.com/, 1997.
[2] The Tcl Plugin Home Page. Published under URL: http://sunscript.sun.com/plugin, 1997.
[3] The Java Home Page. Published under URL: http://java.sun.com/, 1997.
[4] S. Jablonski and C. Bussler. *Workflow Management Modeling Concepts, Architecture and Implementation.* Thomson Computer Press, 1996.
[5] K. Kozminski. Design Control for a Silicon Compiler. Technical Report 1991-TR@MCNC, MCNC, P.O. Box 12889, Research Triangle Park, RTP, NC 27709, January 1991. This report is now available as a postscript file under http://www.cbl.ncsu.edu/publications.
[6] (Ed.) K. Kozminski. OASIS 2.0 User's Guide. MCNC, Research Triangle Park, N.C. 27709, 1992. (Over 600 pages, distributed to over 60 teaching and research universities worldwide).
[7] D. S. Harrison, R. A. Newton, R. L. Spickelmier, T. J. Barnes. Electronic CAD Frameworks. *Proceedings of IEEE*, 78(2):1062–1081, February 1990.
[8] J. Daniell and S. W. Director. An Object Oriented Approach to CAD Tool Control Within a Design Framework. *IEEE Transactions on Computer-Aided Design*, 10(6):698–713, June 1991.
[9] A. Casotto and A. Sangiovanni-Vincentelli. Automated Design Management Using Traces. *IEEE Transactions on Computer-Aided Design*, 12(8):1077–1095, August 1993.
[10] J. B. Brockman and S. W. Director. The Schema-based Approach to Workflow Management. *IEEE Transactions on Computer-Aided Design*, 14(10):1445–1267, October 1995.
[11] O. Bentz, J. M. Rabaey, and D. Lidsky. A Dynamic Design Estimation and Exploration Environment. In *Proceedings of the 34th Design Automation Conference*, pages 190–195, June 1997.
[12] H. Lavana, A. Khetawat, F. Brglez, and K. Kozminski. Executable Workflows: A Paradigm for Collaborative Design on the Internet. In *Proceedings of the 34th Design Automation Conference*, pages 553–558, June 1997. Also available at http://www.cbl.ncsu.edu/publications/.
[13] J. K. Ousterhout, J. Y. Levy, and B. B. Welch. The Safe-Tcl Security Model. Published under URL: http://www.sunlabs.com/people/john.ousterhout/SafeTcl.ps, March 1997. Draft.