

**WebWiseTclTk, OmniDesk and OmniFlows:  
A User-Configurable Distributed Design  
Environment inside a Web-Browser**

Hemang Lavana Franc Brglez

April 1998

1998-TR@CBL-03-Lavana, Version 1.0

**Reprinted, with permission, from**  
<http://www.cbl.ncsu.edu/publications/>

Publications at this site are occasionally revised,  
please check for the latest version under the same title.

**For more information about CBL, visit**  
<http://www.cbl.ncsu.edu/>

**or write to**  
[info@cbl.ncsu.edu](mailto:info@cbl.ncsu.edu)

©1998 authors and CBL, All Rights Reserved

## ABOUT THIS DOCUMENT

**Acknowledgments.** The work presented in this document has been supported by by contracts from the Semiconductor Research Corporation (94-DJ-553), SEMATECH (94-DJ-800), and DARPA/ARO (P-3316-EL/DAAH04-94-G-2080 and DAAG55-97-1-0345), and a grant from Semiconductor Research Corporation.

**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 a toolkit `WebWiseTclTk` and demonstrates a range of its applications in support of a heterogeneous computing environment where CAD tools, data, and users are distributed across the continent. Specifically, we implemented *OmniDesk*, as a Tcl 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, tool-specific, and data-dependent tasks. Identical versions of *OmniDesk* and a variety of user-configurable *OmniFlow* class libraries can be mirrored on several Web sites or can be installed locally for faster access and execution.

The `WebWiseTclTk` toolkit is an enhancement of the existing feature set of `Safe-Tcl` and `Safe-Tk`, that does not compromise security. The toolkit re-defines the functionality of the `auto_load` mechanism in Tcl such that it works for packages located anywhere on the World Wide Web. It also re-introduces several commands not available in `Safe-Tk` such as `toplevel` and `menu` to provide a much richer feature set of Tk commands. The toolkit is written entirely in `Safe-Tcl/Tk` and uses the *home policy* for running applications as Tcl-plugins.

The toolkit supports (1) creation of new Web-based Tcl applications with greatly enhanced functionality, and (2) migration of existing Tcl applications to the Web by merely writing an encapsulation script. For example, only a simple encapsulation script is required for Web-based execution of the *Tk Widget Demonstrations*, distributed with the core Tcl/Tk.

**Keywords.** Internet, desktops, frameworks, Web browsers, html-forms, CGI scripts, applets, security.

**Note.** This document has been published for viewing on the Web in Postscript and HTML formats. The latter is annotated with active hyperlinks to facilitate quick browsing of this and related documents.

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

```
@techreport{
1998-TR@CBL-03-Lavana,
author = "H. Lavana and F. Brglez",
title = "{WebWiseTclTk, OmniDesk and OmniFlows:
        A User-Configurable Distributed Design
        Environment inside a Web-Browser}",
institution = "{CBL, CS Dept., NCSU, Box 7550,
              Raleigh, NC 27695}",
number = "1998-TR@CBL-03-Lavana",
month = "April",
year = "1998",
note = "{Also available at
        {\tt http://www.cbl.ncsu.edu/publications/}}"
}
```

## CONTENTS

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>II</b>	<b>Background on WebWiseTclTk</b>	<b>1</b>
<b>III</b>	<b>Motivation</b>	<b>2</b>
<b>IV</b>	<b>OmniDesk – User View</b>	<b>4</b>
<b>V</b>	<b>OmniFlow – User View</b>	<b>5</b>
<b>VI</b>	<b>OmniFlow Scheduling and Execution</b>	<b>6</b>
<b>VII</b>	<b>Applications and Experiments</b>	<b>7</b>
<b>VIII</b>	<b>Conclusions</b>	<b>8</b>

## LIST OF FIGURES

1	Architecture for WebWiseTclTk toolkit. . . . .	3
2	Dependency graph, OmniFlow and OmniDesk. . . . .	4
3	OmniDesk and OmniBrowser within a Web-browser. . . . .	5
4	OmniFlow depicting the partitioning, logic optimization, placement and routing. . . . .	5
5	Expanded view of a hierarchical OmniFlow node. . . . .	5
6	HTML-form for executing partitioner. . . . .	6
7	Program node configuration. . . . .	6
8	Data and script node configuration. . . . .	6
9	Firing schedule of an executable Petri net. . . . .	7
10	Cell-based placement and evaluator OmniFlow. . . . .	7

## WebWiseTclTk, OmniDesk and OmniFlows:

## A User-Configurable Distributed Design Environment inside a Web-Browser

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 a toolkit *WebWiseTclTk* and demonstrates a range of its applications in support of a heterogeneous computing environment where CAD tools, data, and users are distributed across the continent. Specifically, we implemented *OmniDesk*, as a Tcl 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, tool-specific, and data-dependent tasks. Identical versions of *OmniDesk* and a variety of user-configurable *OmniFlow* class libraries can be mirrored on several Web sites or can be installed locally for faster access and execution.

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

## I. INTRODUCTION

The problems and solutions proposed in this paper are addressed in the context of developers and users of technology that supports the design of microelectronic systems. Increasingly, this technology 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 technology of WWW to create a platform-independent user-reconfigurable and scalable CAD environment that can support the design needs of distributed teams on a global scale.

This paper introduces a toolkit *WebWiseTclTk* and demonstrates a range of its applications in support of a heterogeneous computing environment where CAD tools, data, and users are distributed across the continent. Specifically, we implement *OmniDesk*, as a Tcl 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,

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).

“Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of CBL. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.”

© 1997 CBL

to *OmniFlow* applets that can execute any sequence of user-defined, tool-specific, and data-dependent tasks. Identical versions of *OmniDesk* and a variety of user-configurable *OmniFlow* class libraries can be mirrored on several Web sites or can be installed locally for faster access and execution.

The *WebWiseTclTk* toolkit is an enhancement of the existing feature set of *Safe-Tcl* and *Safe-Tk*, that does not compromise security. The toolkit re-defines the functionality of the `auto_load` mechanism in Tcl such that it works for packages located anywhere on the World Wide Web. It also re-introduces several commands not available in *Safe-Tk*, such as `toplevel` and `menu`, to provide a much richer feature set of Tk commands. The toolkit is written entirely in *Safe-Tcl/Tk* and uses the *home policy* for running applications as Tcl-plugins.

Since both the *OmniDesk* and any *OmniFlow* are executed as applets in *WebWiseTclTk* [1, 2], users need to install a Tcl-plugin [3, 4] in their browser. Implementing either or both as applets in Java [5] is feasible but not necessary. Our decision to extend Tcl/Tk [6, 7] to *WebWiseTclTk* is based on favorable experience with the Tcl/Tk reported in the CAD tool supplier community.

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*.

Formalized descriptions of workflow modeling concepts, architecture, and implementation are still evolving, e.g. [8]. Many are application-specific rather than generic. Our approach expands on the attributes of the design control language *decol* [9] 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 [10]. Some aspects relate to the work in the area of design methodologies and design frameworks such as [11, 12, 13, 14].

The paper is organized into the following sections: (2) background on *WebWiseTclTk*; (3) motivation; (4) user view of the *OmniDesk* environment; (5) user view of the *OmniFlow* environment; (6) highlights of *OmniFlow* scheduling and execution; (7) summary of current applications; and (8) conclusions.

## II. BACKGROUND ON WebWiseTclTk

A large application, written in Tcl, typically consists of a short main script and a library of support scripts. Applications start up quickly by invoking the main script. As new features are accessed, the code that implements them is loaded automatically, using the `auto_load` mechanism available in Tcl.

The Tcl-plugin, based on Safe-Tcl, restricts running such large applications inside a Web-browser. A few of these restrictions are listed below:

- Auto\_load scheme fails, unless the application package is installed on the client host. Another alternative is to merge all the scripts in the application into a single script which can be downloaded as a *tclet*.
- Applications are restricted to a single window since the command `toplevel` is not available in Safe-Tk and new windows cannot be created.
- Menu widgets are also disabled in Safe-Tk.
- *Tclets* do not have access to standard input/output.

The Tcl-plugin supports multiple security policies so that the *tclets* can perform any of the functionality described above. However, this requires every client host to devise and customize their security policies for every application before accessing these as *tclets*.

It is desirable that the Tcl applications be easily translated into *tclets* and made readily available on the World Wide Web:

- without requiring any major changes in the code, and
- without requiring any sophisticated security policy.

We have developed the `WebWiseTclTk` toolkit as an enhancement to the Tcl-plugin that makes use of the *home policy* only. The *home policy* is, by default, enabled in the Tcl-plugin and hence applications using `WebWiseTclTk` do not require the host clients to change their existing security policies.

The toolkit `WebWiseTclTk` consists of two parts: (1) `WebWiseTcl` which is an enhancement for Safe-Tcl and is useful for applications that do not require display, and (2) `WebWiseTk` which is an enhancement for Safe-Tk for applications requiring display. The toolkit itself consists of several smaller scripts and uses the modified `auto_load` mechanism designed for `WebWiseTclTk`.

Figure 1(a) shows the general architecture that implements the `auto_load` mechanism. Special cases of the generalized architecture, shown in Figures 1(b), (c) and (d), are described below:

1. Typically, a client host downloading a *tclet* from a Web server has only the Tcl-plugin installed for its Web-browser. The server site hosts not only the *tclet scripts* but also the `WebWiseTclTk` toolkit as shown in Figure 1(b). The client host downloads the main script for the *tclet* which requests to use the *home policy*. If the client host has not disabled the *home policy*, then the main script downloads the initialization script of the `WebWiseTclTk` toolkit from the server site. Once the initialization has completed, the `auto_load` mechanism is modified to dynamically download the remaining scripts of the application as and when needed during its execution.

2. In the second case, shown in Figure 1(c), the client host has locally installed the `WebWiseTclTk` toolkit. The main script of the downloaded *tclet* uses the locally available toolkit and visits the server site only to retrieve its other scripts. This results in faster execution of the *tclet* code.

3. In the third case, shown in Figure 1(d), the `WebWiseTclTk` toolkit is neither available on the server site nor is it installed on the client host. It is available at the software repository site. This requires the client host to install a special *WebWiseTclTk policy* that allows the *tclets* to download scripts not only from its server site but to also download the toolkit from our site. This mechanism has the advantage of always using the latest version of the `WebWiseTclTk` toolkit.

The generalized architecture allows the main *tclet* script to dynamically use one of the above three mechanisms, based on the configuration of the client host. Full details about `WebWiseTclTk` are available in [1]. To check out a demo or download `WebWiseTclTk`, see [2].

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. URL-based access to data and tools over the Web has distinct advantages over the existing methods, including:

- It is no longer necessary to create accounts for every remote user planning to access the tool,
- Tools do not have to rely on X-based connections, requiring high bandwidth for broadcasting GUI to the remote user,
- Installation of the software on the remote user's site is not needed, and
- Remote users always access the latest version of the tool.

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 (access may be password controlled);
- 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 to any other program in a specific workflow sequence.

Advantages include:

- simplicity of implementation (for Web-page designers);
- convenience for one-time use (for Web-page users).

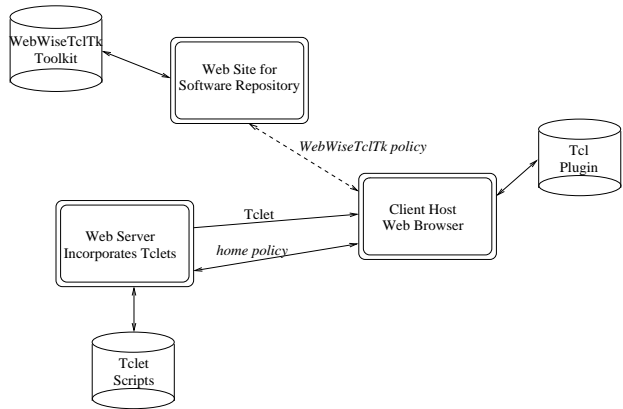
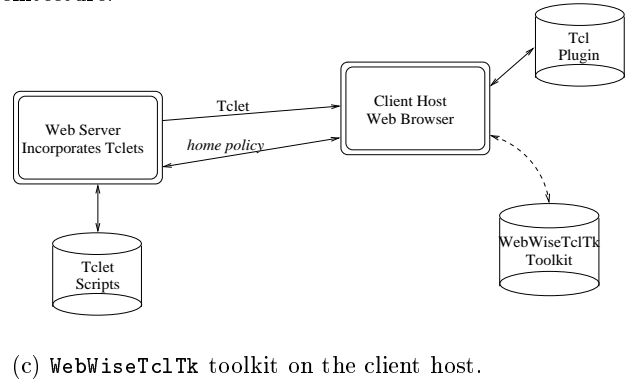
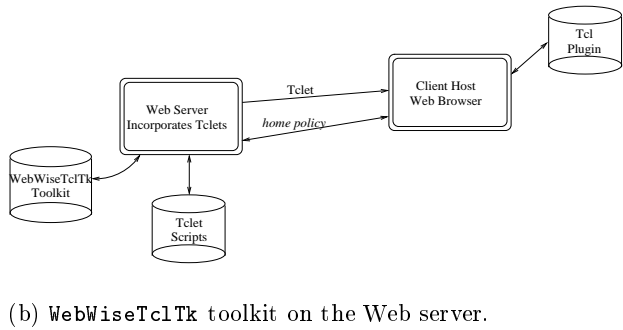
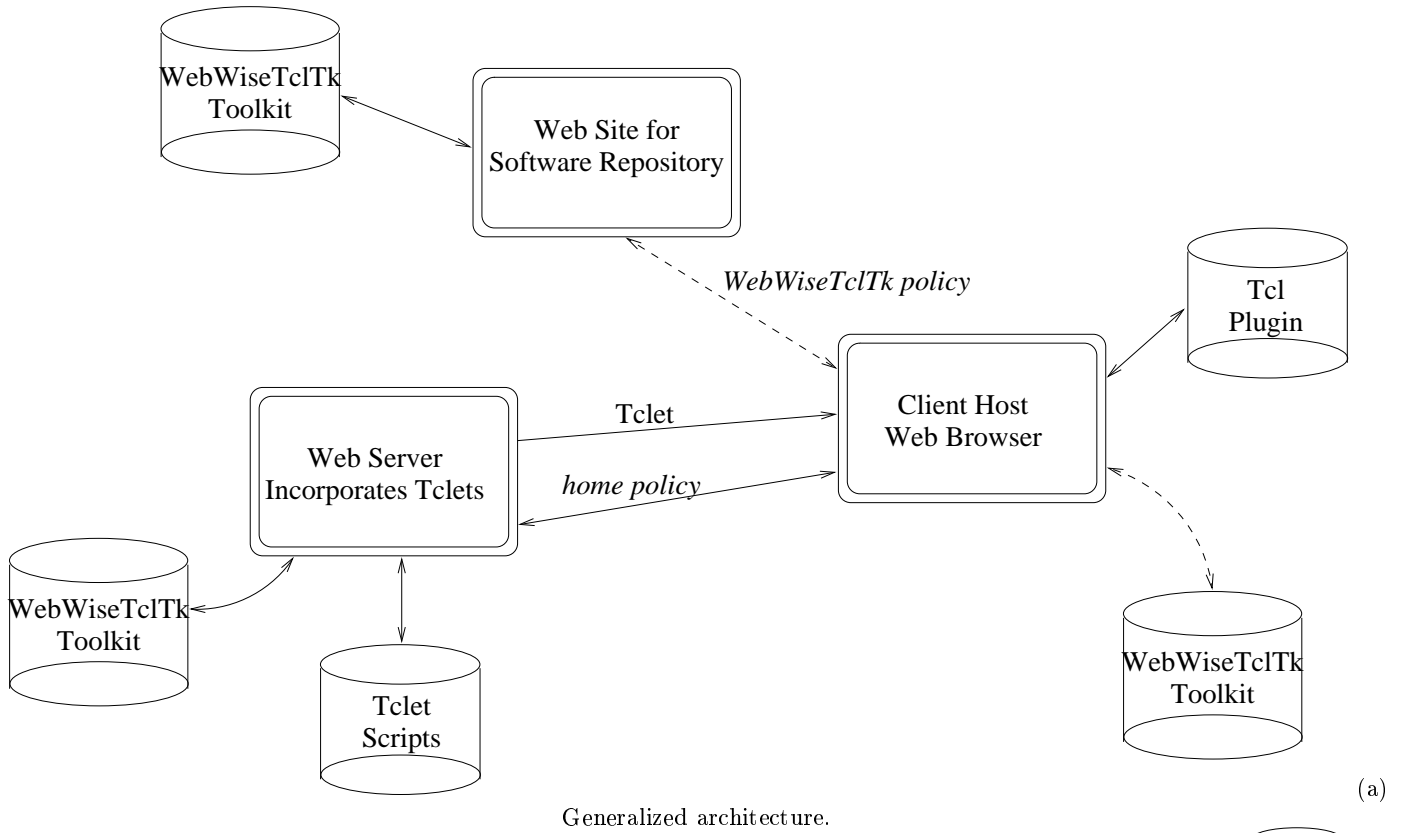
Drawbacks include:

- 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 illustrated in Figure 2(a). Here, we consider the role of three point tools on the Web to support a typical segment of the design process:

- `partitioner@URL4` extracts `init-partition@URL5` from the circuit description `circuit@URL2`;
- `optimizer@URL8` attempts to reduce the size of the initial circuit partition, returning `opt-partition@URL9`;
- `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 2(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



- Typical scenario of a *tcllet* execution:
1. Client host visits the Web server.
  2. Main *tcllet* script is downloaded to client host.
  3. WebWiseTclTk toolkit is loaded and initialized.
  4. Main *tcllet* script executes.
  5. Individual *tcllet* scripts are downloaded as and when required.

Fig. 1. Architecture for WebWiseTclTk toolkit.

iteration. It is always possible to create an executable batch script that invokes a series of single-point tools. However, a high level of programming skill is required for creating such a script. In addition, nominal users cannot re-configure these scripts to suit their needs without the assistance of a skilled programmer.

We argue that distributed users be given the flexibility to configure dependency graphs, such as shown in Figure 2(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 2(b). Here, we depict two users, *user1* and *user2*, engaged in very different tasks. The main objectives of *user1* are:

- to download the applet `OmniDesk@cbl`;
- to *configure* a specific *OmniFlow* that uses programs such as `program2@mit` and data such as `data1@ucb`;
- 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:

- to browse `OmniFlow-lib@mpi` and download a specific *OmniFlow*;
- 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.

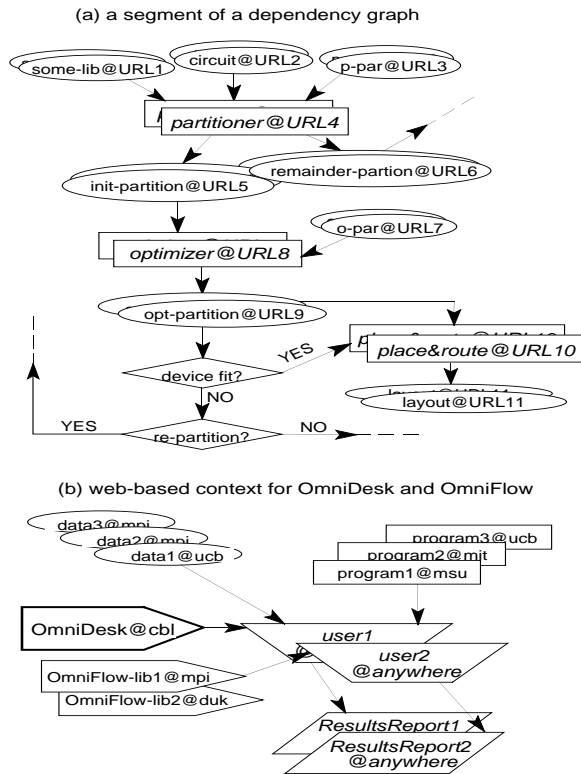


Fig. 2. Dependency graph, *OmniFlow* and *OmniDesk*.

**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 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.

*OmniDesks* and *OmniFlows* can be thought to be analogous to the proxy-based distributed architecture described in [15]. Proxies that implement the role of a conventional legacy tool ‘wrapper’, by hiding a custom interface for a particular tool or class of tools on the client side, are represented in *OmniFlows* as program nodes that provide a uniform interface to the client side for accessing any type of tool – be it a conventional legacy tool or an html-form-based CGI tool. Similarly, *OmniDesk* represents several other services provided by proxies such as data hiding, security management, etc.

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: (1) convenient for repetitive applications, since each form is entered only once and saved as a configuration file; (2) URL for the html form is never accessed during the execution, reducing the network traffic; (3) output data of any Web-based programs are transferred automatically as input data to other Web-based program; (4) *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; and (5) libraries of *OmniFlows*, each executable within the *OmniDesk*, can be created and maintained at any Web-site. Drawbacks include slightly higher setup-time, not suitable 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[6] can then be used and re-used in any number of *OmniFlows*.

#### IV. OMNIDESK – USER VIEW

*OmniDesk* may be invoked by downloading the Tcl applet from a Web-server using 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 3. Ini-

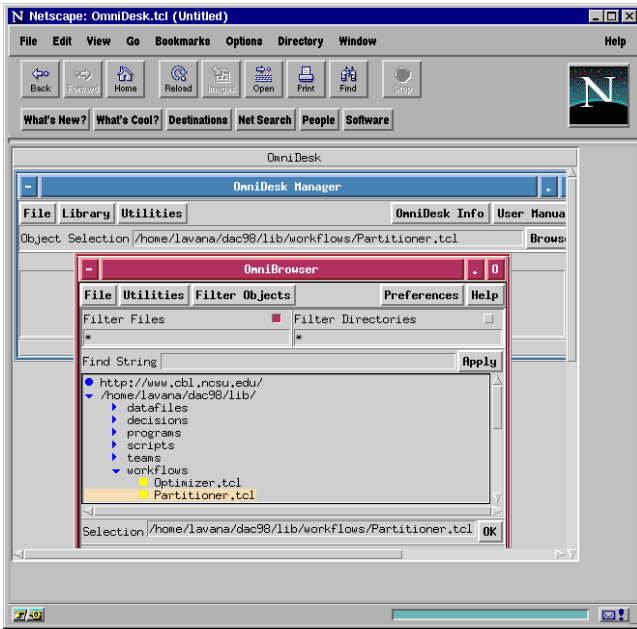


Fig. 3. OmniDesk and OmniBrowser within a Web-browser.

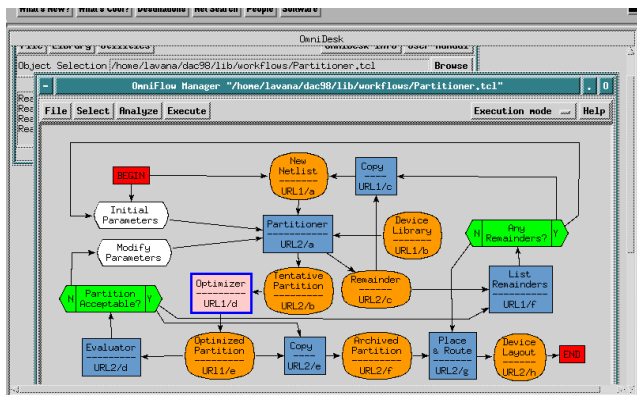


Fig. 4. OmniFlow depicting the partitioning, logic optimization, placement and routing.

tially, it contains only two windows - the OmniDesk Manager and the OmniBrowser.

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 3. On the other hand, when a URL corresponding to a specific Web-site is visited, the contents of the specified URL are 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.

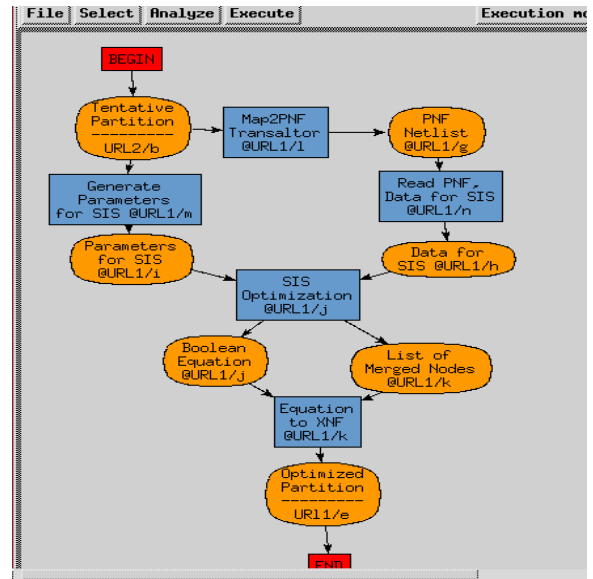


Fig. 5. Expanded view of a hierarchical OmniFlow node.

## V. OMNIFLOW – USER VIEW

Any OmniFlow, such as shown in Figure 4, is an executable graph-based representation of 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 an Optimizer in Figure 4, expand into OmniFlows of their own as shown in Figure 5.

We next describe the steps involved in creating program and data node configurations to execute a CGI program and contrast it with the 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 6 shows an 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.

Figures 7 and 8 show an equivalent representation of the html-

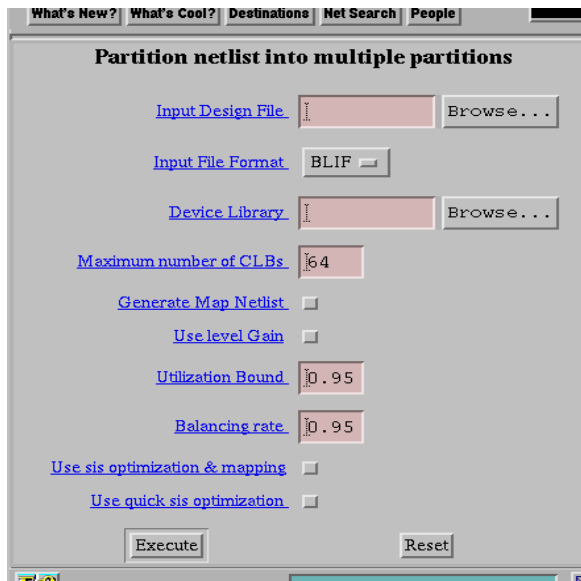


Fig. 6. HTML-form for executing partitioner.

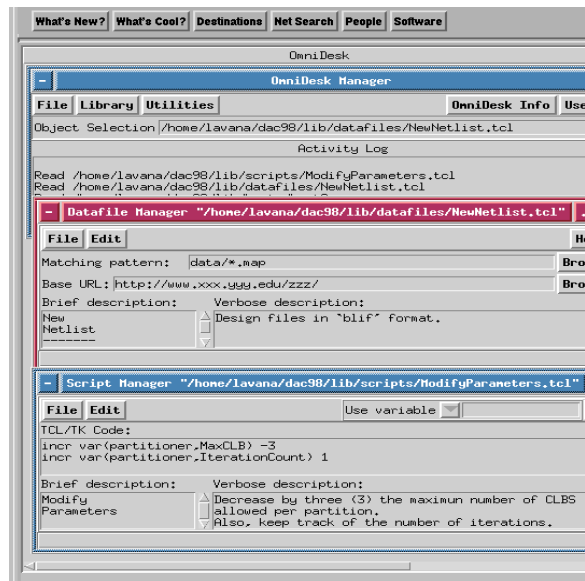


Fig. 8. Data and script node configuration.

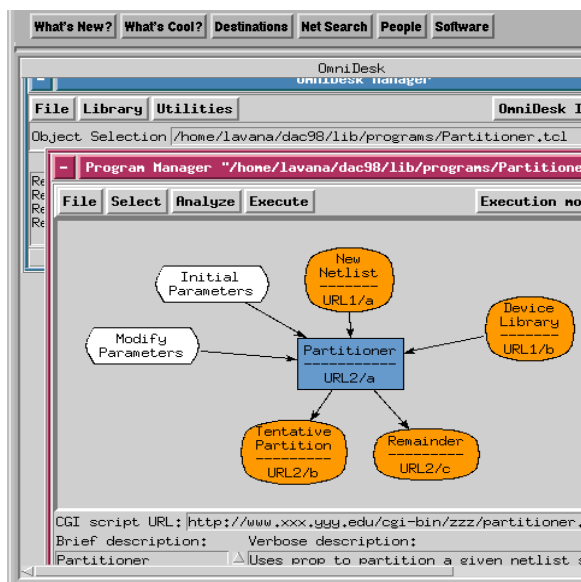


Fig. 7. Program node configuration.

form-based entry as program node configuration and data node configuration templates. To create a program template, such as in Figure 7, 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 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 8 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 the 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, provide 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 8 shows a window for specifying a script node template Modify Parameters that decreases 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.

## VI. 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 adopt the formal Petri Net based approach 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.

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. 9. Firing schedule of an executable 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 9 shows a feasible firing schedule for the OmniFlow shown in Figure 4.

## VII. 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 4 illustrates its potential to customize a *design flow* of distributed tools. It represents one of the Internet-based desktop environments that will be tested as an integral part of a national level collaborative and distributed design project involving teams at several sites.

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.

Figure 10 shows an example of an OmniFlow as a *testbed* to support comparative benchmarking of algorithms during various phases of cell-based placement. The testbed uses the equivalence classes of circuit mutants to support rigorous statistical analysis, including a test of hypothesis such as ‘*Is the improvement due to choice of the algorithm or due to chance?*’

The significant feature of the flow is the collection and evaluation of data generated by any user-selected options that control execution of the flow. The evaluator, `Place_Eval`, is *shared* by all phases of any placement algorithm, and reports on not only area and wire length, but also *wire crossing, critical wire crossing, and critical wire length*, etc. Any number of placement algorithms, located on remote hosts, can be encapsulated into this testbed and evaluated, with all data being archived automatically at a common site for further analysis.

**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

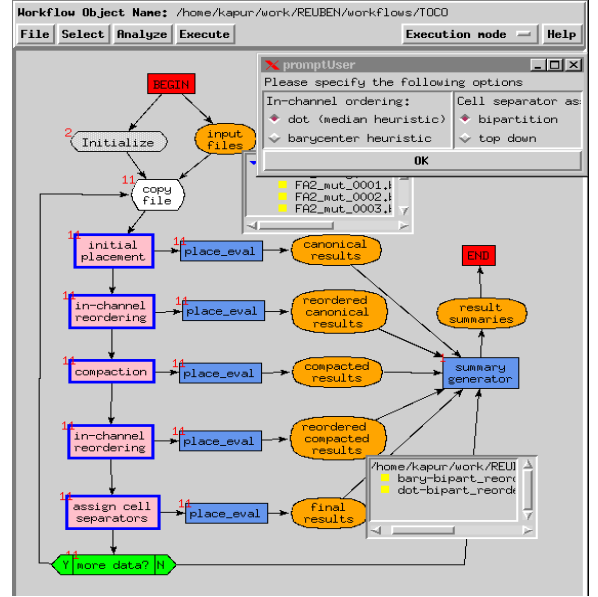


Fig. 10. Cell-based placement and evaluator OmniFlow.

installing the Tcl-plugin on our host<sup>1</sup>; (2) *cross-state environment* by installing the Tcl-plugin on a host<sup>2</sup> at Duke University in Durham, NC; and (3) *cross-country environment* by installing the Tcl-plugin on a host<sup>3</sup> at the University of California in Berkeley, CA.

We 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 4. As shown, the OmniFlow has 20 nodes and 25 arcs; during execution, the node labeled as

<sup>1</sup>SUN SPARC 20 (chip=60MHz memory=64Mb swap=732Mb)

<sup>2</sup>SUN SPARC Ultra 1 (chip=167MHz memory=256Mb swap=288Mb)

<sup>3</sup>SUN SPARC 20 (chip=60MHz memory=96Mb swap=365Mb)

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 the `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` provides information as to 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 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`’ whereas the bottom two report the average values of ‘`user_time`’ and ‘`system_time`’ for each experiment.

**Summary of Results.** The data presented in Table I allows us to evaluate the performance of the OmniDesk environment.

1. The ‘`real_time`’ for the three hosts varies, depending on the characteristics of the host processor and the distance between the OmniDesk Web-site and the host. Specifically, Duke host consistently reported least execution times, followed by our host and UCB host. This is attributed to its higher performance host processor.
2. The minimum and maximum values of ‘`real_time`’ for each experiment, when performed during the night, are close to each other. 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 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.

TABLE I

SUMMARY OF EXPERIMENTS PERFORMED ON THE INTERNET.

Operation	OUR-host		Duke-host		UCB-host	
	Min	Max	Min	Max	Min	Max
Real Time <sup>a</sup>						
CPU Time <sup>b</sup>	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

<sup>a</sup>Both minimum and maximum values of ‘`real_time`’ are reported.

<sup>b</sup>Only average values of ‘`user_time`’ and ‘`system_time`’ are reported.

**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%) to one another – under favorable network conditions and server load.
- Performance of the Web-based OmniDesk environment is quite good under nominal network traffic and server load. Hence, with sufficient network bandwidth and powerful processors, it is possible to work efficiently and effectively on a design project, via the Web-browser, accessing the tools and data dispersed across the continent.

## VIII. CONCLUSIONS

We have demonstrated the capabilities of the WebWiseTclTk toolkit by encapsulating a number of Tcl/Tk applications, including the Tk Widget Demos, as an executable *Tclet* on the Web [2]. In addition, we have proposed and implemented a Tcl-based 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] H. Lavana and F. Brglez. WebWiseTclTk: A Safe-Tcl/Tk-based Toolkit Enhanced for the World Wide Web. Technical Report 1998-TR@CBL-02-Lavana, CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695, Apr 1998. Also available at <http://www.cbl.ncsu.edu/publications/>.
- [2] The WebWiseTclTk Toolkit Home Page. The software is available at <http://www.cbl.ncsu.edu/software/#WebWiseTclTk>.
- [3] The Tcl Plugin Home Page. Published under URL: <http://sunscrip.tcl.com/plugin>, 1997.
- [4] J. K. Ousterhout, J. Y. Levy, and B. B. Welch. The Safe-Tcl Security Model. Published under URL: <http://scrip.tcl.com/people/john.ousterhout/safeTcl.ps>, March 1997. Draft.
- [5] The Java Home Page. Published under URL: <http://java.sun.com/>, 1997.
- [6] The Tcl/Tk Home Page. Published under URL: <http://sunscrip.tcl.com/>, 1997.
- [7] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.
- [8] S. Jablonski and C. Bussler. *Workflow Management Modeling Concepts, Architecture and Implementation*. Thomson Computer Press, 1996.
- [9] 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>.
- [10] (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).
- [11] D. S. Harrison, R. A. Newton, R. L. Spickelmier, T. J. Barnes. Electronic CAD Frameworks. *Proceedings of IEEE*, 78(2):1062–1081, February 1990.
- [12] 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.
- [13] A. Casotto and A. Sangiovanni-Vincentelli. Automated Design Management Using Traces. *IEEE Transactions on Computer-Aided Design*, 12(8):1077–1095, August 1993.
- [14] 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.
- [15] M. Spiller and R. Newton. EDA and the Network. In *International Conference on Computer Aided Design*, pages 470–476, Nov 1997.