

**OmniDesk/OmniFlow: An Agent-Based Architecture and a Toolkit  
for Building Configurable and Collaborative Workflows of Heterogeneous  
Applications and Data for the Web**

Hemang Lavana   Franc Brglez   Robert Reese   Gangadhar Konduri

October 1998

1998-TR@CBL-10-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 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.** The increasing accessibility of heterogeneous 'point tools' on the Web raises the question on how to best encapsulate such tools in order to create (1) simple-to-use workflows of task-specific tools and data, (2) communication channels between distributed participants to coordinate the collaborative control and execution of the workflow until all pre-defined tasks are completed. This paper describes an agent-based architecture (OmniDesk) and a toolkit (OmniFlow) that support Web-based user-configurable encapsulation environment and a GUI to create domain-specific, *distributed and collaborative workflows* of heterogeneous applications and data, all executable within a Web-browser.

The project is driven by two complementary drivers and is beginning to demonstrate the promise, the capability, and the extensibility of OmniDesk/OmniFlow environment: (1) as a testbed for distributed design of collaborative experiments for performance evaluation of CAD algorithms, (2) as a testbed for distributed, re-configurable, and collaborative workflows of university-based and commercial tools to support a distributed design team in a major VLSI design project. Preliminary experiments demonstrate that the performance of the proposed Web-based collaborative architecture is nearly independent of the cross-continental distribution of participants.

**Keywords.** Design environments, workflows, Web-browsers, applets, security, collaboration, collaborative agents.

**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-10-Lavana,
author = "H. Lavana and F. Brglez and R. Reese and G. Konduri",
title = "OmniDesk/OmniFlow:
        An Agent-Based Architecture and a Toolkit
        for Building Configurable and Collaborative Workflows
        of Heterogeneous Applications and Data for the Web",
institution = "{CBL, CS Dept., NCSU, Box 7550,
              Raleigh, NC 27695}",
number = "1998-TR@CBL-10-Lavana",
month = "October",
year = "1998",
note = "{Also available at
\{\tt http://www.cbl.ncsu.edu/publications/\#1998-TR@CBL-10-Lavana\}"
}
```

## CONTENTS

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>II</b>	<b>WISIWYS and NISNYS Paradigms</b>	<b>2</b>
<b>III</b>	<b>Collaborative Architecture (OmniDesk)</b>	<b>3</b>
<b>IV</b>	<b>Reconfigurable Workflow (OmniFlow)</b>	<b>4</b>
<b>V</b>	<b>Examples of Embeddable Web-Based Applications</b>	<b>6</b>
<b>VI</b>	<b>Collaborative Experiments on the Web</b>	<b>7</b>
<b>VII</b>	<b>Conclusions</b>	<b>8</b>

## LIST OF FIGURES

1	Graph-based representation and its GUI for a hierarchical and distributed workflow and its states, executed as a collaborative session of three teams. . . . .	2
2	Architecture for collaboration on the Web. . . . .	3
3	Tcl code to setup agent for collaboration. . . . .	4
4	Graduated netlist mutation OmniFlow. . . . .	5
5	Configuration template for a program node. . . . .	5
6	Configuration template for a data node. . . . .	5
7	JavaCADD: Mentor layout tool task window. . . . .	6
8	Execution of PowerPlay from within WebTop. . . . .	6
9	Execution of a Synopsys-Partitioner-Xilinx flow. . . . .	6
10	Collaborative experiment on the Web with 'Solitaire'. . . . .	7

## LIST OF TABLES

I	Summary of experiments: events executable per second . . . . .	8
---	----------------------------------------------------------------	---

# OmniDesk/OmniFlow: An Agent-Based Architecture and a Toolkit for Building Configurable and Collaborative Workflows of Heterogeneous Applications and Data for the Web

Hemang Lavana, Collaborative Benchmarking Lab, Dept. of CS, Box 7550, NC State U., Raleigh, NC 27695.  
Franc Brglez, Collaborative Benchmarking Lab, Dept. of CS, Box 7550, NC State U., Raleigh, NC 27695.  
Robert Reese, MPL/MSU, Engr. Research Center, P.O. Box 6176, Mississippi State U., Starkville, MS 39762.  
Gangadhar Konduri, Dept. of EECS, Massachusetts Institute of Technology, Cambridge, MA 02139.

**Abstract** – *The increasing accessibility of heterogeneous ‘point tools’ on the Web raises the question on how to best encapsulate such tools in order to create (1) simple-to-use workflows of task-specific tools and data, (2) communication channels between distributed participants to coordinate the collaborative control and execution of the workflow until all pre-defined tasks are completed. This paper describes an agent-based architecture (OmniDesk) and a toolkit (OmniFlow) that support Web-based user-configurable encapsulation environment and a GUI to create domain-specific, distributed and collaborative workflows of heterogeneous applications and data, all executable within a Web-browser.*

*The project is driven by two complementary drivers and is beginning to demonstrate the promise, the capability, and the extensibility of OmniDesk/OmniFlow environment: (1) as a testbed for distributed design of collaborative experiments for performance evaluation of CAD algorithms, (2) as a testbed for distributed, re-configurable, and collaborative workflows of university-based and commercial tools to support a distributed design team in a major VLSI design project. Preliminary experiments demonstrate that the performance of the proposed Web-based collaborative architecture is nearly independent of the cross-continental distribution of participants.*

**Keywords:** Design environments, workflows, Web-browsers, applets, security, collaboration, collaborative agents.

## I. INTRODUCTION

The ubiquity of the World Wide Web and the Internet, with its client/server model and the telecommunication protocols, are stimulating developments of large number of diverse applications, all accessible and executable through a user-friendly Web-browser interface. In its simplest form, the interface provides the user with an option to either enter the required data through a form, upload it from a local file system, or a combination of both. Upon completion of execution, the user is provided with answers on-line, a URL to download data to the local file system, or a combination of both. More elaborate system applications may provide menus for interactive entry of graphical objects, menu-driven controls to execute a workflow of applica-

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 and DAAG55-97-1-0345), and a grant from Semiconductor Research Corporation.

“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.”

© 1998 CBL

tions, and multiple choices of output data from each of the executed applications. However, there is currently no supportive environment for putting together a *user-configurable hierarchy* of executable workflows, given the heterogeneous applications available on the Web. In addition, most of such applications have been designed for interactions with a single user rather than a distributed team.

With the increasing accessibility (on the Web) of heterogeneous ‘point tools’, workflows, and teams with *expertise* in using these tools, the question arises on how best to encapsulate such tools in order to create (1) simple-to-use workflows of task-specific tools and data, (2) communication channels between distributed participants to coordinate the collaborative control and execution of the workflow until all pre-defined tasks are completed. Architectures addressing items (1) and (2) were proposed in [1, 2, 3]; while effective, these architectures are not suitable for the current Web technology. An environment addressing (1) only, in context of the Web, is described in [4]. This paper describes an architecture (OmniDesk) and a toolkit (OmniFlow) that support Web-based user-configurable encapsulation environment and GUI to create domain-specific, *distributed and collaborative workflows* of heterogeneous applications and data, all executable through a Web-browser.

Since the proposed architecture and the toolkit are to support an encapsulation environment for any number of heterogeneous programs and data, a scripting language is a first candidate for the task [5]. Our choice of TclTk [6] as the implementation language for OmniDesk/OmniFlow has been influenced by a number of factors:

- the large and active community of developers worldwide, backed by a commitment from a tool vendor to continued free distribution of Tcl core and its updates [7];
- the accounts of mission-critical Tcl applications from industry such as the complete NBC’s broadcast automation system [8] and the increasing number of TclTk applications in EDA industry;
- the security mechanisms for applets available with Safe-Tcl plugin [9] but not yet with Java [10], such as fine-grained access and configurable security policies;
- the introduction of the TclTk extension *WebWiseTclTk* [11, 12], significantly expanding the features of Safe-Tcl plugin in current Web-browsers. Using WebWiseTclTk, developers can now ‘wrap’ existing multi-window Tcl-based applications for immediate Web-access as tclets – providing full capabilities of the original application but without rewriting the existing Tcl code and without compromising the security of the existing Safe-Tcl plugin;
- the emphasis on support for collaborative implementation and the diversity of applications, few of which are likely to be compliant with CORBA [13].

We are experimenting with two complementary project

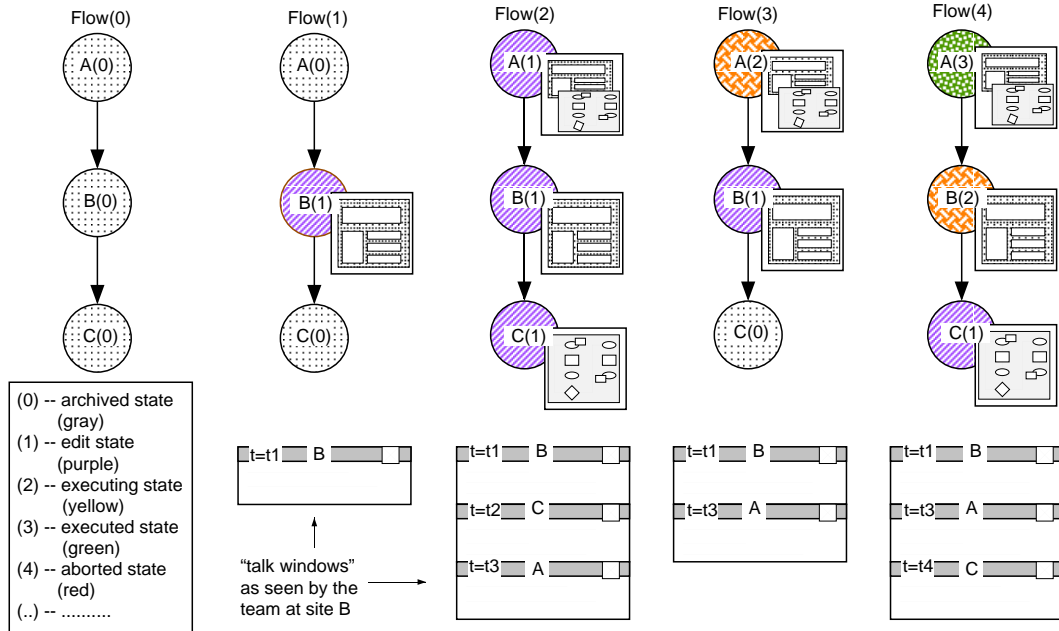


Fig. 1. Graph-based representation and its GUI for a hierarchical and distributed workflow and its states, executed as a collaborative session of three teams.

drivers to demonstrate the promise, the capability, and the extensibility of OmniDesk/OmniFlow environment: (1) a testbed for distributed design of collaborative experiments for performance evaluation of CAD algorithms, (2) a testbed for distributed, re-configurable, and collaborative workflows of university-based and commercial tools to support a distributed design team in a major VLSI design project. In this paper, we highlight the tool-integration experiences with (2) from a perspective of a distributed, university-based team. Experiences with (1) are described elsewhere [14].

The paper is organized into following sections: (2) WISIWYS and NISNYS Paradigms, (3) Collaborative Architecture (OmniDesk), (4) Reconfigurable Workflow (OmniFlow), (5) Embeddable Web-Based Applications and Flows, (6) Collaborative Experiments on the Web, and (7) Conclusions.

## II. WISIWYS AND NISNYS PARADIGMS

WISIWYS is an acronym for *What-I-Sce-Is-What-You-Sce* [15]. It describes a collaboration paradigm that provides distributed session participants with identical displays of an application. Moreover, an interaction by any user with the application is presented to all session participants simultaneously. To date, WISIWYS is limited in support for applications within the Web-browsers such as Netscape conferencing tools. In this paper, we introduce an architecture that supports WISIWYS for Tcl-only applications, executed through the Web-browser. Such browser must have an installed Safe-Tcl Plugin 2.0.

NISNYS is an acronym for *Now-I-Sce-Now-You-Sce*. It describes a collaboration paradigm that allows participants to exchange views and control execution of any single-user legacy (non-Tcl) application serially, either through the Web-browser or without. The paradigm supports a number of interactive collaborative activities and leverages the features of the proposed WISIWYS application in TclTk. We describe an approach to implement the WISIWYS/NISNYS paradigm in the next sections. The purpose of this section is to motivate and illustrate

the underlying features of both paradigms.

**Illustrative design problem.** Let  $DP$  be a design problem that requires, for an effective solution, three toolsets and three teams at locations  $A$ ,  $B$ , and  $C$ , each one or more time-zones away from the other. Labels associated with location  $A$  include:  $team\_A$ ,  $toolsets\_A$  and  $datasets\_A$  on  $host\_A$ ,  $time-stamps\_A$ ; similar labels are created for locations  $B$  and  $C$ . All toolsets are executable through a Web-browser as described in the preceding section: they can be Java or Tcl applets, or CGI-scripts executing legacy applications. Data dependencies between the toolsets are as follows: given the initial  $datasets\_A$ , latest results of computations on  $host\_A$  are required to initiate and complete computations on  $host\_B$ , latest results of computations on  $host\_B$  are required to initiate and complete computations on  $host\_C$ . If results of computations on  $host\_C$  do not meet design specifications, the process may be repeated on either the  $host\_B$  by adjusting parameters in  $datasets\_B$ , or the  $host\_A$  by adjusting parameters in  $datasets\_A$ . Such decisions are made in consultation with team members from all sites. We argue that the WISIWYS/NISNYS paradigms can greatly facilitate and improve the collaborative decision-making process in the context of  $DP$ .

Let  $HG$  be a graphical user interface that supports creation of hierarchical directed multi-graphs: (multiple) edges from node  $A$  to node  $B$  are represented as a single super-edge and by clicking on node  $A$  or  $B$ , the node is expanded either into a set of configuration forms that describe properties of the node, or another directed multi-graph until we reach a set of configuration forms for each node. A typical configuration form for a node of type 'data' will identify its class by the extension (e.g. \*.txt, \*.html), and point to the owner, host, and the directory where all data (with time-stamps) in this class is archived or can be up-dated. Similarly, a configuration form for a node of type 'program' will contain an encapsulation script, the owner, the host, and the directory where it is to be executed, with pointers to required/optional input/output data nodes. Time-stamp dependencies and a Petri-net model of the multi-graph formalize

conflict-free scheduling and execution of all program nodes, even in the presence of data-dependent loops. We call the complete, executable representation of such a multi-graph a *workflow*.

We use the graphical user interface *HG* to represent the dependencies and the flow of execution for the design problem *DP*. At the top level of *DP* representation, we have three nodes,  $(A, B, C)$ , and two directed superedges,  $((A, B), (B, C))$ . The corresponding workflow, in its ‘archived state’, is shown in Figure 1 under the label *Flow(0)*. Each of the nodes, *A*, *B*, and *C*, now represents stand-alone workflow of its own. However, some of the output data generated on host *host\_A* is designated as ‘required input’ to applications on node *B*; and similarly, some of output data generated on host *host\_B* becomes the ‘required input’ to applications on node *C*.

**Collaborative execution.** Assume that the workflow *Flow(0)* is archived on a *collaboration Web-server* and can be checked-out by an authorized member of team *A*, *B*, or *C*. The *collaborative execution* of this workflow can be captured in a number of *flow states* and *node states* as illustrated in Figure 1 under the labels of *Flow(1)*,  $\dots$ , *Flow(4)*:

*Flow(1)*: Here, *team\_B* checks out *Flow(0)* and initiates an editing session by clicking on node *B*. While view of the 3-node workflow is available, only clicking on node *B* produces a response since other teams have R/W permission to workflows associated with other nodes. Three more transactions take place: state of flow changes to *Flow(1)*, state of node *B* changes to *B(1)* (purple), and a ‘talk window’ is opened, with the status bar marking the entry time ( $t = t_1$ ), the window owner (*B*), and a checked permission status box to indicate that by default, only *team\_B* has presently R/W permission at this node.

*Flow(2)*: Here, *team\_C* has joined the session at time  $t = t_2$ , followed by *team\_A* at time  $t = t_3$ . States of all nodes are (1), indicating that all teams are in editing mode. There appear to be some written communication between team members through the respective talk windows.

*Flow(3)*: We observe that node *A* is in state *A(3)* (yellow), hence executing, while *team\_C* has left the session (node *C* is returned to state *C(0)* and the talk window for *C* has been removed).

*Flow(4)*: Here, node *A* has completed execution which allowed for execution of node *B*. Next, *team\_C* has re-joined the session at time  $t = t_4$  with expectation to be in a position to initiate the execution of node *C* once it become noted that the node *B* has change to state *B(2)*.

We discuss the role of WISIWYS and NISNYS in the context of the collaborative session as just described. In order for the session to be ‘user-friendly’, the workflow and its states must be continually visible, to *all* session participants, i.e. WISIWYS must be supported for at least the top level of hierarchy of the workflow. Once WISIWYS is achieved for the Tcl-application at the top-level, any Tcl-application invoked at the top-level is also available as WISIWYS. However, rendering a legacy (non-Tcl) EDA application collaborative as WISIWYS through a Web-browser is an open research problem. The alternative NISNYS paradigm can be realized more readily in the context of our current work:

- team controlling the legacy application and associated data requests technical assistance from another team. To get assistance, the team must be willing to relinquish control by unchecking the checked permission status box in the team’s ‘talk window’;
- once the permission status box signifies permission to access the top-level node, another team can exercise control over the legacy application, view the input data and render the output,

say circuit layout, and then advise on a course of action. The control can change any number of times, to resolve the outstanding issues.

The challenge then is to devise an architecture to support collaboration as well as a toolkit to allow users to build their own workflows on the Web.

### III. COLLABORATIVE ARCHITECTURE (OMNIDESK)

Architectures supporting distributed, collaborative applications are subject to open research, in particular applications that are to be executed through an off-the-shelf Web-browser. Traditional pre-web architectures that support collaboration [16] either rely on centralized server model on which all applications reside, or a replicated model where applications are distributed and reside on the local host of each session participant. None of these models are suitable for the development of Web-based collaborative applications. The centralized model is impractical due to heterogeneity of Web clients. On the other hand, replicated architecture may become too complex to manage, in addition to unresolved security issues.

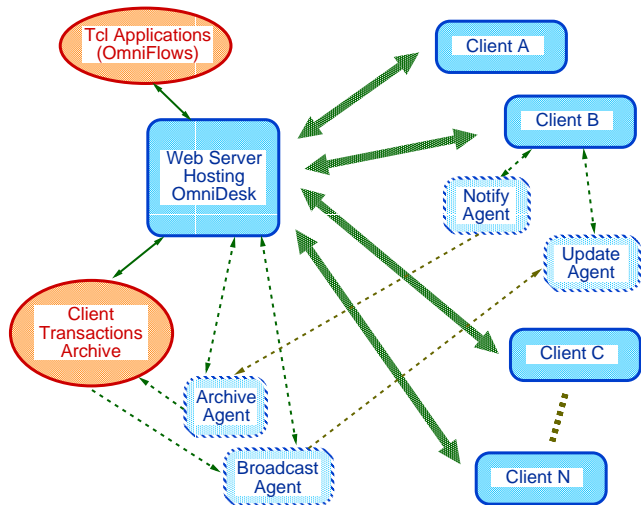


Fig. 2. Architecture for collaboration on the Web.

The powerful plug-ins for the Web, such as Safe-Tcl and its extensions, WebWiseTclTk, can support a new *hybrid architecture*, adopted for our Web-based WISIWYS/NISNYS experiments, as shown in Figure 2. NSTP [17] also uses a hybrid approach, however, our architecture targets Web-based applications. There is a dedicated Web-server hosting a collaborative Tcl application *OmniDesk*, a library of other Tcl applications, notably *OmniFlows* which is another name for workflows to be described in the next section, and *Client Transaction Archive*. Clients are *Tcl-plugin enabled* Web-browsers *A*, *B*, and *C* associated with *team\_A*, *team\_B* and *team\_C*. For the context of this discussion, we assume that the Tcl applications such as *OmniFlows* have been designed for at least 2 collaborating clients and the task of *OmniDesk* is to set-up and manage the WISIWYS/NISNYS functionality as described in the previous section. For each client, a *Notify Agent* and an *Update Agent* are set-up automatically by *OmniDesk* once the client requests a Tcl application, say *Foo*, from its *OmniDesk* library. The first instantiation of the *Notify Agent* also instantiates the *Archive Agent* and the *Broadcast Agent* on the Web-server host. These two agents have responsibility to archive any state changes of the Tcl application (*Foo*) induced by the first and later participants, and to broadcast these state changes to all Update

Agents activated by clients who requested the application Foo. In other words, while there can be up to  $n$  preauthorized users requesting to checkout the application Foo (and subsequently, up to  $n$  Notify and  $n$  Update Agents), there is only one Archive Agent and one Broadcast Agent associated with the application Foo.

Consider the workflow application in Figure 1 to illustrate the role of the OmniDesk in support of the WISIWYS/NISNYS environment.

Whenever the client  $B$  tries to access an workflow from the Web server, such as one shown in Figure 1 Flow(0), the OmniDesk is also downloaded to the client. Client  $B$  is initialized with a workflow that contains all its nodes to be in archived state and hence shaded gray. At this time, no other client is accessing the same workflow and therefore, client  $B$  is set up for interaction with the flow in single user mode. However, the OmniDesk also creates two agents: the Notify Agent, and the Update Agent, which contact the Web server to establish a live connection with the Archive Agent and the Broadcast Agent residing on the Web server respectively. The role of the Notify Agent is to transmit any user interaction with the workflow to the Archive Agent on the Web server, which in turn logs the transaction. At this time, the Update Agent is idle since there are no other clients participating in the workflow execution. The OmniDesk also sets up a talk window for client  $B$ , as shown in Figure 1 Flow(1), that can be used for communication with other participants that may join the session later. Thus, as soon as the client  $B$  starts interacting with her node, such as editing data, the node turns purple and the transaction is logged in the archive.

Now, let us assume that client  $C$  wants to join the collaborative session. It downloads the OmniFlow and the OmniDesk and undergoes a similar process of setting up agents and creating a talk window as client  $B$  did earlier. In addition, the OmniFlow of client  $C$  is initialized to the current state of client  $B$  by retrieving  $B$ 's transactions from the archive. This results in rendering the  $B$  node in the workflow to be purple. At the same time, client  $B$  is informed by the broadcast agent that a new participant, client  $C$ , has joined the session. The update agent of client  $B$  then creates a talk window for client  $C$  to receive messages typed by client  $C$ . Once client  $C$  starts interacting with the workflow, these transactions are also logged in the archive and simultaneously broadcast to client  $B$ .

In a similar fashion, any number of clients can join an existing collaborative session. Figure 1 Flow(2) shows three participants  $A$ ,  $B$ , and  $C$  editing or executing different nodes of the same workflow. It also shows three talk windows, one corresponding to each participant. The top-left corner of the talk window also displays the time at which each participant has joined the session.

An active participant also has the option of leaving a collaborative session simply by closing the OmniFlow/OmniDesk application. The Notifying Agent informs the Archiving Agent about the removal of the participant, which is then broadcast to all other participants. In Figure 1 Flow(3), client  $C$  has left the session. Therefore its talk window is removed from the client  $B$ 's application and the node corresponding client  $C$  is marked gray. Later, client  $C$  joins the collaborative session again, as shown in Figure 1 Flow(4).

A simplified version of Tcl code is shown in Figure 3 to setup and initialize the Update Agent and establish a connection with the Broadcast Agent. The first two lines identify the server URL and setup a security policy for enabling the Agent to establish connection with the Agents on the server. The third line creates

```
# Get the server URL for tclet's server site
set servUrl [getattr originHomeDirURL]
# Set up 'home' policy to connect to server
policy home

# Initiallize Update Agent to connect to
# Broadcast Agent
after idle \
    "::browser::getURL ${servUrl}BroadcastAgent.tcl \
    {} {} UpdateAgent UpdateAgent"

# A procedure to update the application
# on receiving new data
proc UpdateAgent {size data} {
    uplevel #0 $data
}

# Rest of the tclet code ...
```

Fig. 3. Tcl code to setup agent for collaboration.

an Update Agent which always runs in the background. Whenever a transaction is broadcast by the Web Agent, the Update Agent calls an update procedure to bring the main application to a consistent state.

#### IV. RECONFIGURABLE WORKFLOW (OMNIFLOW)

An OmniFlow 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 Web. 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 expand into OmniFlows of their own during execution.

A simple OmniFlow for *Graduated netlist mutation* is shown in Figure 4. Graduated netlist mutation is a tool that is accessible as cgi-script on the Web using an html-form. It expects the user to upload a netlist file, and supply few parameters to generate a set of mutant circuits. The OmniFlow in the Figure 4 depicts a program node *Graduated Mutation* corresponding to the cgi-script and its input and output data nodes. Whereas a user is required to manually fill out the html-form and download the results whenever a netlist data is submitted for execution, the OmniFlow provides sufficient capabilities to:

- represent all the netlists as a data node *Input Netlist*,
- automatically submit each netlist to the cgi-script 'graduated mutation' on the Web, and
- download the resultant mutants generated on invoking the cgi-script.

We next describe the steps involved in creating program and data node configurations to execute a cgi-script. A program node configuration template, which encapsulates a cgi-script,

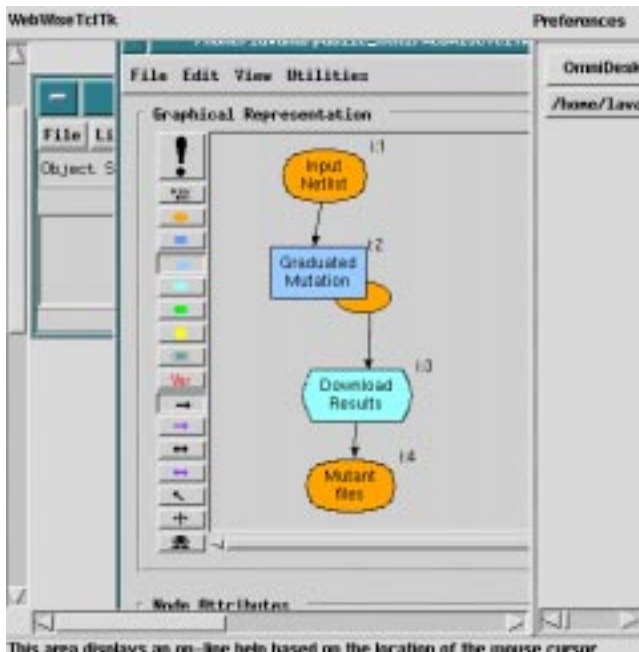


Fig. 4. Graduated netlist mutation OmniFlow.

can be either created by the user, or can be generated automatically from its corresponding html-form.

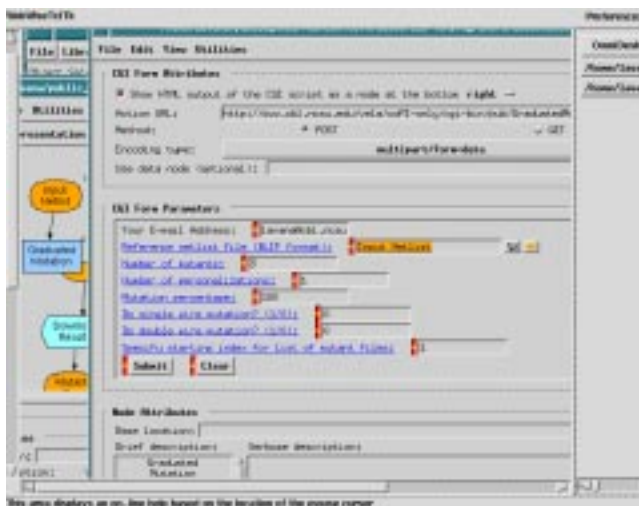


Fig. 5. Configuration template for a program node.

Figure 5 shows the configuration template for the program node *Graduated Mutation*. This node has been instantaneously created by specifying the URL of the corresponding html-form. The top part of the template consists of the various cgi-form attributes such as action URL, encoding method, etc, for invoking the cgi-script. Traditional Web browsers, such as Netscape and Internet Explorer, normally hide this information and display only the cgi-form parameters to the user. In a cgi-script node, we display both the information to the user to enrich the capabilities that are very useful for repeated invocation of the cgi-script. Each parameter in the form is tagged with a star '\*' that allows the user to define a variable corresponding to the parameter. This variable can be dynamically changed during the execution of the workflow, thereby providing a mechanism

to invoke the same cgi-script with modified parameters. Additionally, form-parameters, which expect the user to upload data, can be associated with data nodes to represent a class of data. In Figure 5, the parameter 'Reference Netlist File (blif format)' is linked with the parameter 'Input Netlist'. In such a case, the cgi-script is invoked as many times as there are datafiles ( netlists ) in the data node directory.

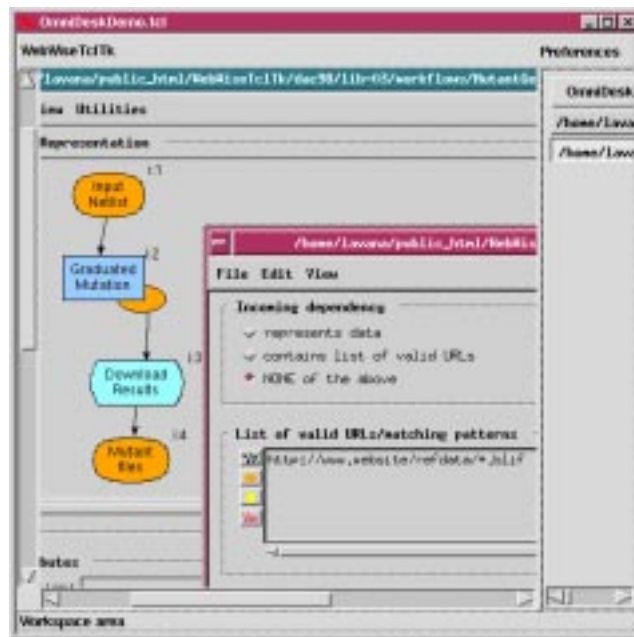


Fig. 6. Configuration template for a data node.

Figure 6 shows the configuration template for the data node *Input netlist*. The most important element of the data node is the location of the data set, specified by a URL, followed by a file-matching pattern. There can be a list of such URL patterns although only one such pattern is shown in Figure 6.

Other cgi-scripts 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.

There are several *custom* client/server architectures besides Web-based cgi-scripts. *Java Remote Method Invocation* (JavaRMI) is a recent addition to such custom client/server architecture that support

- operating system independence, and
  - a sophisticated data packing mechanism to pass complex objects between client/server with minimum programmer effort.
- JavaCADD [18] is a Java-based server and client that relies on JavaRMI for providing user access to batch-oriented, distributed, ECAD services such as synthesis, place/route, etc, without having to provide full login-access for the users. The JavaCADD GUI uses templates stored at the server for GUI definition, so new services can be added without modifying the Java source code. Figure 7 shows the client GUI for accessing and executing the services of Mentor layout tool using JavaCADD. This mechanism has several distinctive advantages, including:
- full login-access not required,
  - easy addition of new services without modifying client GUI,
  - automated transfer of input and output files between the client and the server,

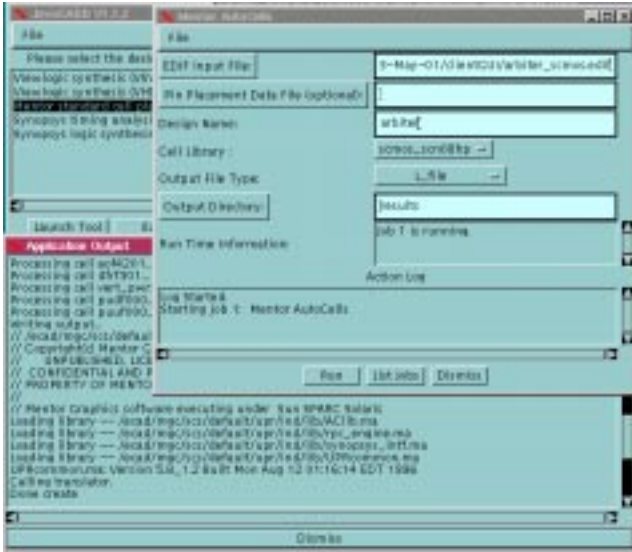


Fig. 7. JavaCADD: Mentor layout tool task window.

- operating system independence, etc.
- OmniFlows can easily encapsulate specific tasks of JavaCADD client GUI, such as one shown in Figure 7, as a program node. The configuration template of a program node for JavaCADD is very similar to the one shown in Figure 5:
- the action URL is specified as `jcadd://servername:[port]/taskPath/taskName` instead of `http://servername:[port]/cgi-bin/script`
  - the cgi-form parameters are replaced by parameters for the JavaCADD specific task.
- Similarly, several other applications can be encapsulated in the OmniFlow. The next section describes few such Web-based applications.

#### V. EXAMPLES OF EMBEDDABLE WEB-BASED APPLICATIONS

Experiments are in progress with a number of distributed web-based applications. These include three sites, *A*, *B*, in *C*, discussed hypothetically in Section 2. To show the heterogeneous nature of tools and interfaces, consider the examples of real applications executable at three remote sites:

- A Java based hierarchical schematic/block editor, WebTop which supports hierarchical cells and multiple views of a cell [19]. WebTop allows cells to have Verilog or SPICE views in addition to the schematic views. WebTop has hierarchical netlisting capabilities for both Verilog and SPICE. WebTop also supports distributed cell access. Cells could be stored in different Web servers and loaded as URLs. WebTop has interfaces to other Web tools (high-level power estimation of media processor, RTL power estimation of video compression, DC-DC converter) and demonstrates how an Java applet tool could utilize other CGI based tools on the Web. An example of the multi-window interface is shown in Figure 8. Here, the user is executing PowerPlay from within the WebTop.
- Distributed design library services and batch-oriented ECAD services with commercial tools. A Java-based server and client architecture allows user access to batch-oriented ECAD services (synthesis, place/route, etc) without having to provide full login-access for the users. The JavaCADD GUI uses templates stored at the server for GUI definition, so new services can be added without modifying the Java source code of the GUI. Java

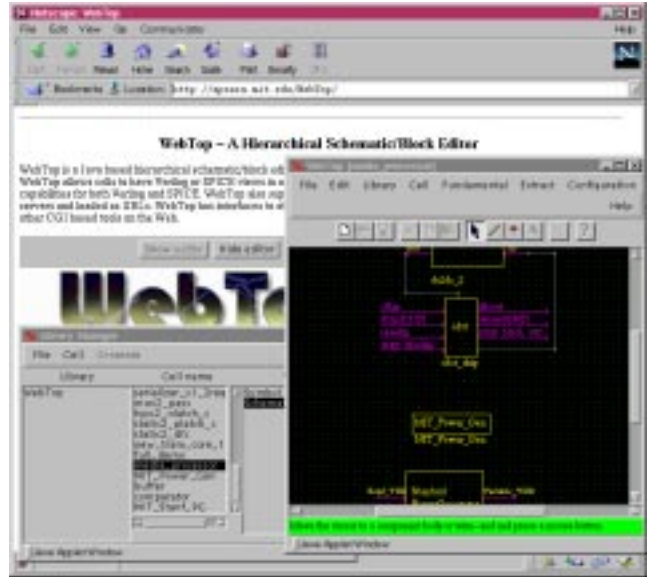


Fig. 8. Execution of PowerPlay from within WebTop.

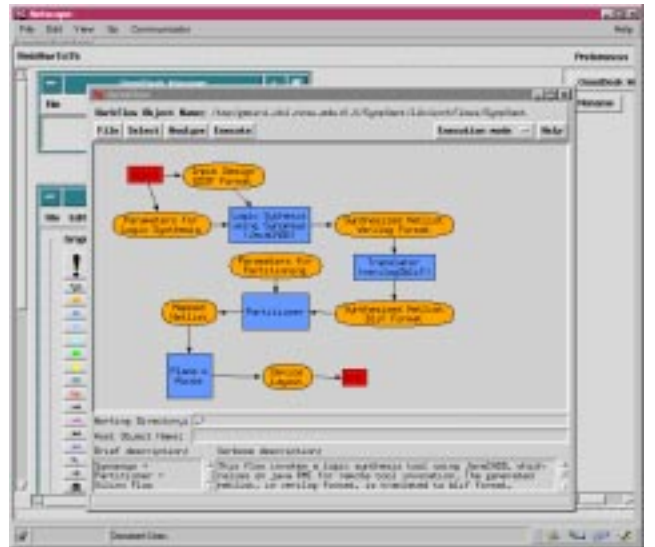


Fig. 9. Execution of a Synopsys-Partitioner-Xilinx flow.

Remote Method Invocation (RMI) is used for client/server communication. Both client and server are Java applications, and share the operating system independence unique to Java. An example of a simple form-based interface is shown in Figure 7.

- A hierarchical, re-configurable desktop environment that provides the user with the ability to create *executable* directed dependency graphs as workflows of data, program, decision, script, and workflow nodes. Data and programs can reside anywhere on the Internet, and execution of all nodes can be scheduled automatically, regardless of the data-dependent cycles in the graph. Such an environment can serve the needs of distributed design teams as well as support an infrastructure for the distributed design of experiments to test performance of CAD algorithms. An example of such a workflow is shown in Figure 9. This particular workflow executes a commercial tool (Synopsys) on a remote site, transfers the netlists to another hosts where it is partitioned, then mapped and placed into multiple FPGA devices using a commercial software (Xilinx).

Additional sites, with similar tools, also part of this project, are participating in creating a number of task-specific distributed workflow configurations, each executable and executed collaboratively in the WISIWYS/NISNYS modes. An example of a realistic application with such tools includes a scenario, suitable for a demo in the University Booth:

- generating, at site *A*, a hierarchical schematic as per Figure 8, performing verification and power estimation and handing off a standard cell netlist for placement and routing at site *B*;
- accepting the netlist from *A* at site *B*, parameterizing the place and route tool as per options available in the GUI similar to one as shown in Figure 7 and executing the place & route tool;
- evaluate an alternative design option by mapping the design from *A* into a set of multiple FPGA devices as per flow in Figure 9.

## VI. COLLABORATIVE EXPERIMENTS ON THE WEB

The key to the efficiency of the collaborative architecture for Web-based WISIWYS and NISNYS is the performance of the the four collaborating agents in Figure 2. This performance will vary, depending on the network traffic, server cpu and the number of participants.

We have developed a testbed to measure the performance of our architecture. In order to approximate typical instance of a collaborative OmniDesk environment, we have used an application, written in Tcl/Tk, that involves lot of graphical interactions with the user. A Tcl/Tk-based game ‘Solitaire’ [20], shown in Figure 10, provides a very suitable test case. It allows, one user at a time, to play the game while others can watch the game in progress.



Fig. 10. Collaborative experiment on the Web with ‘Solitaire’.

A user typically interacts with the application by picking a card from one location and placing it in another location. Several *events* occur during such an interaction, as follows:

1. A user selects a card by clicking on it with the mouse button. The Tcl command for selection of a card is:

```
pile_card_grab .table %x %y
```

The ‘.table’ is the name of widget on which the cards are laid out and ‘%x’ and ‘%y’ correspond to the current mouse cursor location.

2. With the mouse button depressed, the user moves the card to a new location. This involves dragging the card in response

to the mouse cursor movement as follows:

```
pile_card_move .table %x %y
```

3. The card is placed at the desired location by releasing the mouse button. This results in generation of the following Tcl command:

```
pile_card_drop .table %x %y
```

4. However, if the destination location, where the card is dropped, is not a valid location, then the card returns to its original location. The following Tcl command is used for returning the card:

```
pile_card_return .table
```

These events and the corresponding Tcl commands, generated in response to the user interaction with the application, are transmitted and broadcast to all the other participants using the four collaborative agents. A non-zero time delay exists between occurrence of every such event. This time delay is dependent on the way a user interacts with the application. An average user may execute these four events in one second, whereas a fast user may complete it in half the time.

In order to perform a set of repeatable experiments, regardless of the person who may be interacting with the application, we have designed and performed a series of experiments such that they can be repeated under different loading conditions, with multiple *phantom participants* at a number of cross-continental sites. To create a test case of representative size and complexity, we captured the first 1000 events, along with the corresponding time delay intervals, that were generated during the execution of one such session. The test case that captures all events and time delay intervals allows us to repeat the experiment in a number of different environments.

In order to measure the effect of the network on a collaborative session over the Web, we have created: (1) *local environment* by installing the Archive and Broadcast Agents on our host<sup>1</sup>; (2) *in-state environment* by installing the Archive and Broadcast Agents on a host<sup>2</sup> which is about 30 miles from our host; and (3) *cross-state environment* by installing the Archive and Broadcast Agents on a host<sup>3</sup> which is about 3000 miles from our host. Each experiment is repeated for all the three sites. In addition, we vary the number of users participating during each collaborative experiment. We repeat each experiment by doubling the number of users - we start with a single user, then increase the number of users to 2, 4, 8 and 16.

We also simulate the interactions of different users by changing the time delay interval by a constant factor. For the given set of experiments, we decided to repeat the experiment by scaling the time delay intervals with a factor of 1.0, 0.1 and 0.0. Whereas the scaling factor of 1.0 corresponds to the original speed of user interactions with the application, the scaling factor of 0.0 represents the ideal case where the speed of user interactions with the application is maximum, the time delay intervals being reduced to zero.

Table I summarizes results of these experiments, performed on the three hosts, with three scaling factors and the number of clients varying from 1 to 16. Each experiment consisted of executing 1000 events. Entries in the table represent the *rate of event execution*. A value of 7.69 for a single client, on Host-1, with a scaling factor of 1.0, implies that in average, 7.69 events are executed every second. The rate of 7.69 events/second shows

<sup>1</sup>SUN SPARC station 5 (chip=170MHz memory=64Mb swap=432Mb) at North Carolina State University in Raleigh, NC.

<sup>2</sup>SUN SPARC Ultra 1 (chip=143MHz memory=256Mb swap=288Mb) at Duke University in Durham, NC.

<sup>3</sup>SUN SPARC ULTRA 2 (2 chips=200MHz memory=96Mb swap=365Mb) at University of California in Berkeley, CA.

TABLE I  
SUMMARY OF EXPERIMENTS: EVENTS EXECUTABLE PER SECOND

Server location	Scaling factor	Number of clients				
		1	2	4	8	16
Host-1 <sup>a</sup>	1.0	7.69	7.46	7.19	5.85	3.92
Host-2 <sup>b</sup>	1.0	7.63	7.58	7.25	6.21	3.92
Host-3 <sup>c</sup>	1.0	7.69	7.63	7.41	6.62	4.24
Host-1	0.1	55.56	47.62	34.48	14.17	7.58
Host-2	0.1	52.63	47.62	35.71	17.86	7.63
Host-3	0.1	55.56	52.63	40.00	22.22	8.55
Host-1	0.0	200.00	100.00	47.62	20.00	7.75
Host-2	0.0	166.67	100.00	50.00	20.41	7.52
Host-3	0.0	166.67	100.00	58.82	23.26	8.20

<sup>a</sup>Host-1 is located on our site (Raleigh, NC)

<sup>b</sup>Host-2 is located about 30 miles away from our site (Durham, NC)

<sup>c</sup>Host-3 is located about 3000 miles away from our site (Berkeley, CA)

that we engaged a faster than average user to capture this experiment.

The data reported in Table I allows us to evaluate the performance of agent-based collaborative architecture as follows:

1. With original speed of event execution, the rate of execution only decreases to about half even when the number of clients are increased from a single client to 16 clients. This shows great promise for collaborative applications on the Web.
2. As the time delay intervals are reduced to zero, the rate of execution decreases by about half, whenever the number of clients are doubled. The degradation in performance is expected whenever computationally intensive applications are running.
3. The rate execution is more or less constant and independent of where the server hosts are located. This is primarily due to the efficient architecture where minimal amount of data is transferred among different clients for collaboration.
4. A better cpu for the agent server improves the rate of execution, irrespective of the distance, as is noted for host-3 with more than two clients.

## VII. CONCLUSIONS

We have proposed and implemented (1) an *agent-based* architecture (OmniDesk) for collaboration on the Web that supports the WISIWYS and NISNYS paradigms, and (2) a reconfigurable workflow toolkit (OmniFlow) for encapsulating *heterogenous* applications, including *legacy* applications, on the Web. This environment supports user-configurable OmniFlows of distributed data and distributed university/commercial software, each executable within the Web-browser's OmniDesk window.

The preliminary experiments demonstrate the feasibility and advantages of the proposed collaborative architecture: (1) for real-time applications, the performance of our approach is very promising - even for large number of collaborative users, and (2) for highly intensive applications, the performance is acceptable.

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 Kuznar from U. of Ljubljana (Slovenia), and APR from Xilinx Inc.

## REFERENCES

[1] 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/#1997-DAC-Lavana>.

[2] H. Lavana, A. Khetawat, and F. Brglez. Internet-based Workflows: A Paradigm for Dynamically Reconfigurable Desktop Environments. In *ACM Proceedings of the International Conference on Supporting Group Work*, Nov 1997. Also available at <http://www.cbl.ncsu.edu/publications/#1997-GROUP-Lavana>.

[3] A. Khetawat, H. Lavana, and F. Brglez. Internet-based Desktops in Tcl/Tk: Collaborative and Recordable. In *Sixth Annual Tcl/Tk Conference*. USENIX, September 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-TclTk-Khetawat>.

[4] F. Chan, M. Spiller, and R. Newton. WELD - An Environment for Web-Based Electronic Design. In *Proceedings of the 35th Design Automation Conference*, pages 146-152, June 1998.

[5] J. K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. Article in *IEEE Computer Magazine*, March 1998. Also available at <http://scriptics.com/people/john.ousterhout/scripting.html>.

[6] The Tcl/Tk Consortium. Published under URL <http://www.tclconsortium.org/>, 1998.

[7] Scriptics Corporation. Published under URL <http://www.scriptics.com/>, 1998.

[8] S. Angelovich, K. Kenny, and B. Sarachan. NBC's GENesis Broadcast Automation System: From Prototype to Production. In *Sixth Annual Tcl/Tk Conference*. USENIX, September 1998.

[9] The Tcl Plugin Home Page. Published under URL <http://sunscrip.sun.com/plugin>, 1997.

[10] The Java Home Page. Published under URL <http://java.sun.com/>, 1997.

[11] H. Lavana and F. Brglez. WebWiseTclTk: A Safe-Tcl/Tk-based Toolkit Enhanced for the World Wide Web. In *Sixth Annual Tcl/Tk Conference (Best Student Paper Award)*. USENIX, September 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-TclTk-Lavana>.

[12] The WebWiseTclTk Toolkit Home Page. Published under URL <http://www.cbl.ncsu.edu/software/#WebWiseTclTk>.

[13] Jon Siegel. CORBA Fundamentals and Programming. J. Wiley, 1996.

[14] H. Lavana, R. Reese, and F. Brglez. User-Configurable Experimental Design Flows on the Web: The ISCAS'99 Experiments. In *International Symposium on Circuits and Systems*, June 1999.

[15] XMX : A X Protocol Multiplexer. Published under URL <http://www.cs.brown.edu/software/xmx/>, 1990.

[16] S. Greenberg and M. Roseman. *Groupware Toolkits for Synchronous Work*. Computer-Supported Cooperative Work, Trends in Software Series. John Wiley & Sons Ltd. Currently available as Research Report 96/589/09, Dept. of Computer Science, University of Calgary, Calgary, Canada, November.

[17] J. Patterson, M. Day, and J. Kucan. Notification Servers for Synchronous Groupware. In *ACM Conference on Computer Supported Cooperative Work*, pages 122-129, Nov 1996.

[18] D. Linder, R. Reese, J. Robinson, and S. Russ. JavaCADD: A Java-based Server and GUI for Providing Distributed ECAD Services. Technical Report MSSU-COE-ERC-98-07, Microsystems Prototyping Laboratory, MSU/NSF Engineering Research Center, April 1998.

[19] WebTop - A Hierarchical Schematic/Block Editor. Published under URL <http://apsara.mit.edu/WebTop>, 1997.

[20] Tcl/Tk Plugin Example: 'Solitaire' game. Published under URL [http://www.tcltk.com/training/plugin\\_sol.html](http://www.tcltk.com/training/plugin_sol.html), 1995.