

## User-Configurable Experimental Design Flows on the Web: The ISCAS'99 Experiments

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

Robert Reese  
 MPL (Microsystems Prototyping Lab),  
 EE/COEN Department, Box 9571  
 Mississippi State University, Starkville, MS 39762, USA  
<http://WWW.ERC.MsState.Edu/mpl/>

**Abstract** – *The emerging web-based technologies can play an important role to instill the discipline of Experimental Design and make collaborative planning, executing, reporting, and archiving of experiments with core algorithms in CAD a routine. This paper introduces a user-configurable, cross-platform executable encapsulation environment that supports (1) universal access to distributed archives of equivalence classes as input data sets for the experiments; (2) encapsulation of distributed software that implements specific algorithms; (3) cumulative evaluations and archival of posted results.*

**Keywords:** Internet, collaborative computing, design of experiments, circuit equivalence classes, benchmarking.

### 1 INTRODUCTION

The web-based access to public domain data sets for the user-defined benchmarking experiments in CAD is now taken for granted. However, as exemplified in the companion papers introduced in [1], the discipline of Experimental Design does require new toolkits that will make scientific and collaborative planning, executing, reporting, and archiving of experiments with core algorithms in CAD a routine rather than the exception.

Any experimental design at least involves three basic steps: (1) selection of an equivalence class of experimental subjects, eligible for treatments, (2) application of one or more treatment to the same class, and (3) statistical evaluation of effectiveness of each treatment. In CAD, we can consider ‘treatments’ as algorithms applied to data equivalence classes such as netlists (or hypergraphs) that have a number of properties in common: the same or ‘nearly-the-same’ size, distributions of fanins and fanouts, Boolean functionality, Boolean entropy, etc. Figure 1 presents a generic view of such an experimental design workflow.

This paper introduces a user-configurable and executable encapsulation environment that supports, in the context of Figure 1:

- universal access to *distributed* archives of domain-specific equivalence classes that serve as input data sets for the experiments;
- user-encapsulations of *distributed* tools and software required to implement specific algorithms (denoted as *treatments* in this paper);
- cumulative transfers of results of the experiments for domain-specific processing by common evaluator tools and archival into well-defined, peer-accessible *distributed* data structures.

This research was 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).

The primitive of this environment is the configuration

$$\text{dataInput} \Rightarrow \text{Treatment} \Rightarrow \text{dataOutput} \quad (1)$$

which is enumerated in Section 2 for most plausible scenarios of distributed data and software (Treatments). Section 3 introduces the primitive in (1) as a cross-platform executable called **OmniExpress**. With **OmniExpress**, user can readily design, execute, and archive any number of experiments that involve distributed data and treatments. Section 4 outlines the design of the **Ish**, that is invoked for its services by **OmniExpress**, followed by description of **JavaCADD** client/server, in Section 5, that is invoked for complementary services [2]. Section 6 discusses few brief examples of more complex reconfigurable workflows that use multiple instances of **OmniExpress** for more elaborate experimental designs [3, 4], followed by conclusions, in Section 7.

While the approach presented in this paper is guided by considerations of formalized Experimental Design in a distributed environment, its implementation is generic and can just as well serve and support the needs of a distributed collaborative environment for a number of design projects in microelectronic systems design.

### 2 DISTRIBUTED DATA AND TREATMENTS

Consider a simple flow shown in Figure 2(a). It depicts a program **Treatment** which operates on the input data, to generate the treated data as its output. The program is represented by a rectangular node and the data by an oval node. When the input data node contains several data files, we would like to invoke the **Treatment** program as many times as there are data files. Also, the **Treatment** for any data should be invoked only when the input data is newer compared to its output result. Such a flow can be easily implemented using the ‘make’ utility.

However, data as well as **Treatment** are often widely distributed across the Internet. Figure 2(b) through (f) shows several such scenarios where either the **Treatment**, or the data, or both are not available on the user’s local file system. Currently, there is no tool that attempts to resolve these issues. We discuss each scenario next.

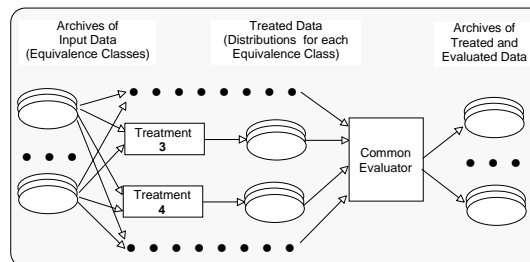


Fig. 1. A generic view of an Experimental Design workflow.

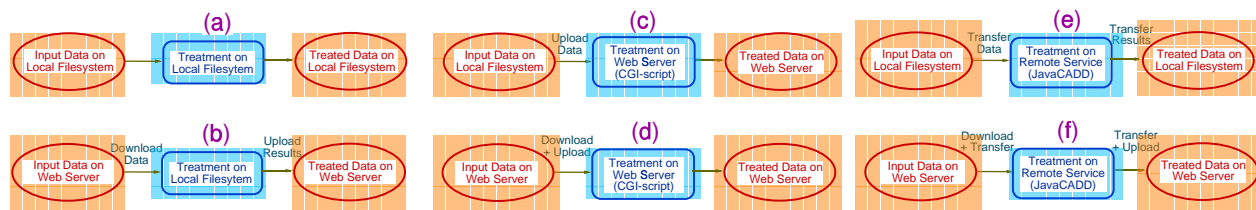


Fig. 2. Examples of invoking various (local as well as remote) treatments with distributed data.

In Figure 2(b), the input data files are readily accessible at a web site. A user would typically download all the data to the local file system using a web browser, apply the **Treatment** locally and post the treated results on a web server by uploading the output files. As the **Treatment** program is improved or debugged, the user is unaware when the new or modified data is available on the web site, unless she verifies the input data every time before running the **Treatment** program. Also, it is very tedious to upload a large number of files, of the order of hundreds or more, to a web site unless the web server accepts upload data in ‘tar.gz’ format.

When the user would like to compare her results with other **Treatments**, it is necessary to access the **Treatment** of peers on a remote site. Figure 2(c) shows a scenario where the **Treatment** is located on a remote site and is accessible as a *cgi-script* on a web server. The user has to upload the input data for **Treatment** and access the treated results from a web site. On the other hand, when the input data is on one web site, and the treatment is on another web server, the user has to first download each data file and then upload it to the web server, as shown in Figure 2(d).

Many times, a **Treatment** is not accessible on a web server, when a web server is not installed, or when the **Treatment** is under development. Login accounts for peer users may not be feasible. Therefore it is necessary to have the treatment accessible on a remote server through a special service, such as JavaCADD, as shown in Figure 2(e) and (f). The first example depicts transfer of data between the remote server and the local file system, whereas the latter depicts downloading and uploading of the data, in addition to the transfer of data.

The aim of this work is to devise an architecture and implement a prototype that can be easily ported and distributed as an executable, thereby allowing the users to: (1) build and execute such simple flows efficiently, and (2) configure more complex flows using these primitives.

### 3 OMNIEXPRESS: IMPLEMENTATION HIGHLIGHTS

We have designed and implemented OmniExpress, an acronym for **Omni-Executable Program Editing and Scheduling System**, in Tcl/Tk due to its robust support for platform independence. OmniExpress is readily compiled into an stand-alone executable that can run on most platforms<sup>1</sup> without requiring installation of Tcl/Tk. The utility to create such executable is part of TclPro [5].

Figure 3 shows a user view of the executable template that encapsulates a **Treatment** as a program node, two data nodes as its input and one data node as its output. For example, the **Treatment**, using SIS, operates on the data file `c1355_mut_0001.blif` with `algebraic` script to generate the resultant data file `c1355_mut_0001_SIS_algebraic.blif`.

<sup>1</sup>Currently supported platforms: Windows 95, NT 4.0 (Intel), Solaris 2.5, 2.6 (SPARC), HP-UX 10.20, Linux (Intel/glibc2, e.g., Red Hat 5.0+, SuSE 5.3+), and SGI IRIX 6.2 to 6.5.

Users can create templates for their own tools by merely editing the flow and configuring the individual nodes. Each node is configured by double clicking on it and filling out a template. Figure 3 shows two such templates - the one on the left corresponds to the data nodes (rd53 and C1355 mutants) and the one on the right corresponds to the **Treatment** that accesses SIS using the JavaCADD remote service. Each node is identified by a unique ‘name’ assigned by the user during its creation. Brief and verbose descriptions aid the user to store additional information about the node - the brief description is also displayed as a text for the node in the flow.

A data node template is a collection of data sets that are located on one or more distributed hosts, including the local file system. Such data is specified by (1) a url or filepath to identify the host, (2) a matching pattern to identify the class of data, and (3) an exclude pattern to eliminate any unwanted data that is matched. For example, the C1355 class D mutants, located on a UCB server, as shown highlighted in the Figure 3, is matched using the pattern `C1355*.blif`. The exclude pattern `c1355*.SIS_algebraic.blif` ensures that the resultant data is not matched.

A program node template also contains the url/filepath attribute to specify where the program resides. In addition, it contains parameters that define the relationship between its input data and output data by use of macros. A macro is of the form `[Nodename::type]` - where each node has several pre-defined types that can be used to dynamically generate its value during execution. For example, the program template in Figure 3, which encapsulates the JavaCADD based SIS **Treatment** on a remote server ‘vela.cbl.ncsu.edu’, shows use of such macros. `[Circuit::filename]` represents the input datafile `c1355_mut_0001.blif` during first invocation, `c1355_mut_0002.blif` during the second invocation, and so on. Several macros can be combined with regular text, such as the one for output filename:

```
[Result::filepath]/[Circuit::fileroot]_SIS_[Script::value].blif
which represents
```

```
c1355_mut_0001_SIS_algebraic.blif,
c1355_mut_0002_SIS_algebraic.blif,
... etc.
```

OmniExpress executes, for the example shown, as follows:

1. compare the time-stamps of all the mutants of rd53 and C1355 from the Duke web site and UCB web site with those of existing output data files,
2. download all the input data from the two web sites that is newer as compared to its corresponding output data,
3. invoke the **Treatment** on ‘vela.cbl.ncsu.edu’ using JavaCADD for each input data by transferring data to ‘vela’.
4. retrieve the output data from ‘vela’

The generic OmniExpress supports a variety of protocols to access remote sites, including telnet, ssh, rsh, etc and relies on Tcl/Tk and Expect [6] for its execution. On the other hand, the stand-alone executable of OmniExpress relies on **Ish** and **JavaCADD**, described in the following sections, to access data and programs on remote sites.

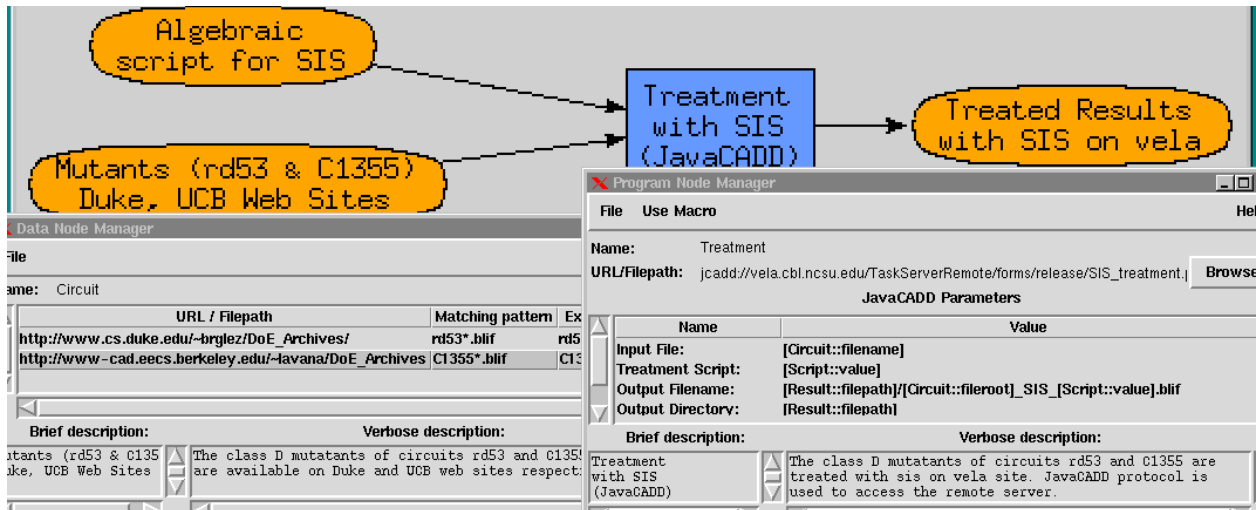


Fig. 3. OmniExpress: executable node with data template and program template.

#### 4 ISH: SHELL-BASED INTERNET CLIENT

Ish, an acronym for **I**nternet **S**hell, is an interactive program that provides a shell interface to easily access web-data and cgi-scripts. Ish is also implemented in Tcl/Tk and is available as a stand-alone executable. Whereas existing web-browsers such as Netscape and Internet Explorer provide an excellent user-interface to access web-data, Ish has been designed to operate directly from the command-line and has built-in support for batch oriented web-access.

In the context of OmniExpress, Ish provides a variety of well-defined services, as follows:

- *Data node listing:* It queries the specified web site and returns a list of data files filtered with the matching pattern. It can also query the time-stamp of any data file.
- *Data node viewing:* It downloads the specified data file and saves it on the local file system.
- *Program node configuration:* It queries the html-form, residing on the web server hosting the cgi-script, and dynamically generates a program template corresponding to the html-form.
- *Program node execution:* It invokes the cgi-script by submitting the required data to the http server with appropriate data encoding as necessary.
- *Data node archival:* It invokes the appropriate cgi-script for uploading the resultant data.

Thus, OmniExpress can encapsulate cgi-scripts and programs existing on the local file system and dynamically generate a program template accordingly. This allows the users to seamlessly integrate web-based data and cgi-scripts along with the local data and programs to provide an infrastructure that implements and executes transparently all the scenarios described in the earlier section.

#### 5 CUSTOM JAVACADD REMOTE SERVICES

Web based services, using html-forms and cgi-scripts, form an excellent mechanism for providing access to programs without giving full login-access. Its limitations include:

1. It is necessary to install a web-server on every machine that intends to host such a service, and
2. Access to local file system on client-side is limited for security reasons.

JavaCADD is a custom client/server architecture [2], based on Java remote method invocation (RMI), which offers so-

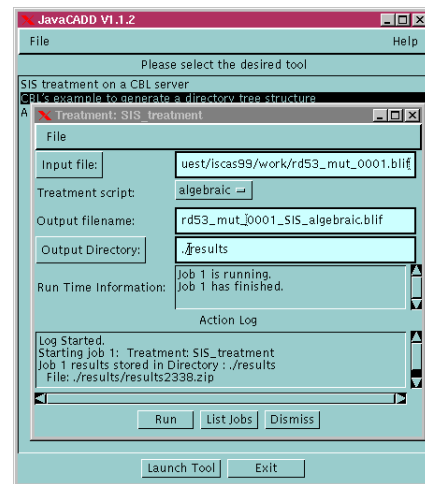


Fig. 4. JavaCADD client GUI.

phisticated data packing mechanism for passing complex objects between the client/server with minimum programmer effort. Both the server and the client implementation of JavaCADD are portable because of Java's operating system independence.

**JavaCADD server.** The function of the server is to accept task requests from client, perform some action to resolve the task, and then return the results to the client. Usually, the task request is a request to spawn an external tool under the control of a user specified shell script but can also include executing a dynamically loaded Java class method or other internal server functions.

A new tool can be easily added as a service by merely creating a configuration file. Configuration files are used to define each tool and the parameters required from the client-side for its execution. Each parameter is a key/value pair and has tight restrictions as to what characters are allowed in its value. For security, a task is rejected whenever any parameter with unallowed character in its value is passed by the client.

**JavaCADD client.** The function of the client is to query the JavaCADD server, on startup, for the list of tool services available and create a GUI for launching the tool.

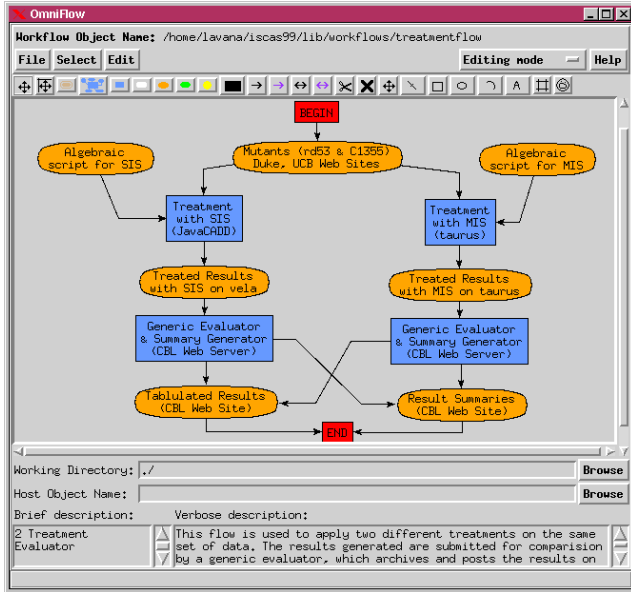


Fig. 5. Reconfigurable workflow for two treatments.

Column	Population Mean	Sample Mean	Sample Std. Dev.	Sample Min.	Sample Max.
Parameter (95% conf. interval)					
Node	[4.0271e+01, 4.1167e+01]	4.0719e+01	2.5377e+00	3.7000e+01	4.6000e+01
Level	[7.2018e+00, 7.4076e+00]	7.3047e+00	5.8267e-01	6.0000e+00	8.0000e+00

Fig. 6. Statistical summary 'rd53\_MIS\_algebraic.sum'.

Upon launching a tool, it again queries the server for the required parameters and dynamically creates a tool specific GUI. Figure 4 shows launch window and a tool specific window for invoking SIS treatment. On running the tool, the input file `rd53_mut_0001.blif` is transferred to the remote server, treated with SIS and the output file is transferred back to the client and saved in the `results` directory.

## 6 RECONFIGURABLE WORKFLOWS USING OMNIEXPRESS

OmniExpress allows us to conveniently perform a *single* task repeatedly. On the other hand, several OmniExpress nodes can be encapsulated in Reuben environment [3] to construct a workflow which performs a series of such tasks. Figure 5 shows a workflow that uses two different Treatments (SIS and MIS) to operate on the same input data. SIS treatment is accessed via JavaCADD, whereas MIS treatment is accessed on a remote host using telnet. The results of the treatments are then submitted to a common evaluator, accessible on the CBL web server, to generate and post summaries. A statistical summary of posted results for 128 Class D mutants of the circuit 'rd53', treated with MIS algebraic script, is shown in Figure 6.

Figure 7 shows yet another application-specific workflow to implement a complex algorithm for the placement scheme - TOCO. It highlights several other features of the Reuben environment, such as decision nodes and hierarchical nodes, necessary to construct a complex workflow. TOCO workflow has been implemented by Nevin Kapur during his MS research and is described in detail in [4].

## 7 CONCLUSIONS

We have discussed the need for a formalized distributed design flow environment in the context of Experimental

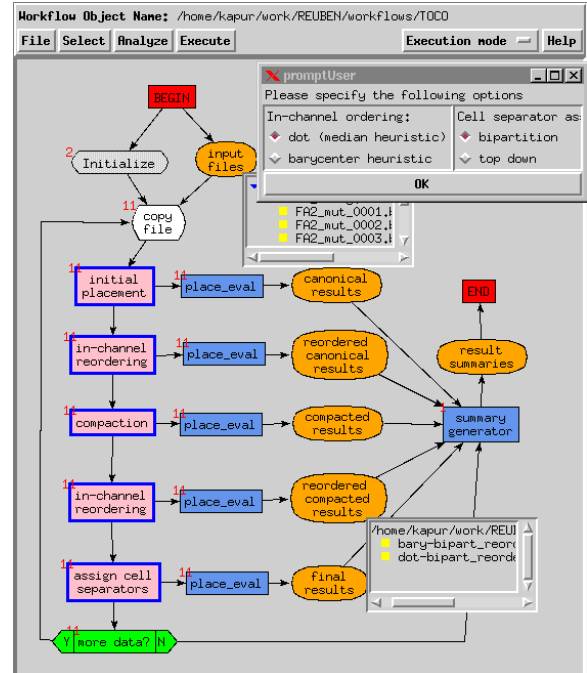


Fig. 7. TOCO: application-specific workflow.

Designs. We have developed and implemented a prototype that (1) supports user-configurable design flows (OmniExpress) for distribution among peer users, (2) a custom Tcl/Tk based Internet shell (Ish) for access to web-data and cgi-scripts, and (3) a custom Java-based server/client architecture (JavaCADD) for access to remote programs of peer users.

The series of ISCAS'99 experiments have provided an ideal driver to demonstrate the capabilities, and the extensibility of the proposed design environment. For updates on experiments and software accessibility, see

[http://www.cbl.ncsu.edu/experiments/DoE\\_Archives](http://www.cbl.ncsu.edu/experiments/DoE_Archives)

**ACKNOWLEDGMENTS.** Several workflows have been constructed and tested by Nevin Kapur and Debabrata Ghosh. We appreciate the access to remote servers and tools used in this research: web servers for hosting data, facilitated by Dr. Richard Newton at UC Berkeley and Dr. Gershon Kedem at Duke U.

## REFERENCES

- [1] F. Brglez and R. Drechsler. Design of Experiments in EDA: Context and New Data Sets for ISCAS'99. In *IEEE 1999 International Symposium on Circuits and Systems - ISCAS'99*, May 1999. A reprint also accessible from <http://www.cbl.ncsu.edu/publications/#I999-ISCAS-Brglez>.
- [2] 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. Also available at <http://WWW.ERC.MsState.Edu/mpl/publications/papers/javacadd/JCaddTR.pdf>.
- [3] 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>.
- [4] N. Kapur. Cell Placement and Minimization of Crossing Numbers. Master's thesis, Electrical and Computer Engineering, North Carolina State University, Raleigh, N.C., May 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-Thesis-MS-Kapur>.
- [5] Scriptics Corporation. Published under URL <http://www.scriptics.com/>, 1998.
- [6] D. Libes. *Exploring Expect*. O'Reilly and Associates, 1995.