1.0 Introduction

Today's collaborative computing environments do not address collaboration using multiple applications executing concurrently. There are many example of systems that allow collaboration using a single application on a single data repository. For examples, HP's SharedX [3] product allows sharing of an X protocol based application among users in a distributed computing environment. MMConf [4] provides the Diamond Multimedia conferencing System. BBN's Slate [5] allows users to collaborate on document development. However, all the above conferencing systems are limited to sharing a central copy of an application. There is no mechanism to concurrently control more than one application or more than one data repository. The missing piece is the subject of this presentation.

We have developed a single server, multiple client environment. ESP, Event Sense Protocol[1], allows a user to control several clients (applications) simultaneously. A key feature of our system is that any existing application can be used with no modification of any kind. For example, ESP enables us to update multiple copies of a Lotus spreadsheet by entering the commands once. In fact, the applications being controlled need not be running on machines of the same architecture or even be identical applications. ESP allows us to control Lotus running on HP and Sun workstations and Excel on an IBM system by typing commands once. The only requirement is that the commands typed be meaningful to each system.

This kind of multiple application, multiple data repository collaboration has many uses. A corporate financial officer could update divisions' independently held spreadsheets to reflect changes in Federal tax law and leave the local revenue numbers unchanged; division heads may change local numbers concurrently. A secretary could change the boiler plate part of standard letters used by all the Company's branch offices.

Combining the conventional single client, multiple server collaborative environment with the single server, multiple client model of ESP completes the picture by providing for multiple application collaboration. Application experts in different locations could simultaneously control various aspects of a distributed application running on several machines. An instructor teaching FrameMaker could control many student documents simultaneously to illustrate a particular point while still allowing the students to work on their own reports and commenting on their content. Systems support people could work with users to improve distributed applications.

The next section describes ESP, a mechanism to enable concurrent application control. Section 3 shows how ESP can be integrated into a collaborative computing environment. Section 4 adds a couple of extended examples of multi-user, multi-application collaboration. Section 5 discusses related work and Section 6 presents our summary.

Internal Accession Date Only

1. Patent pending

2.0 ESP — A Mechanism to Enable Concurrent Application Control

The current graphical user interfaces GUIs [7] are based on a single-threaded dialog, where the user operates on one single command button to invoke one application and to execute one single function at a time. There has been a need to have a mechanism to sense user commands and dialogs, then to control, manage, and multicast them to a set of applications for executing some functions simultaneously. ESP provides this mechanism without requiring any modification to existing system or application software in a distributed environment.

ESP has the following features:

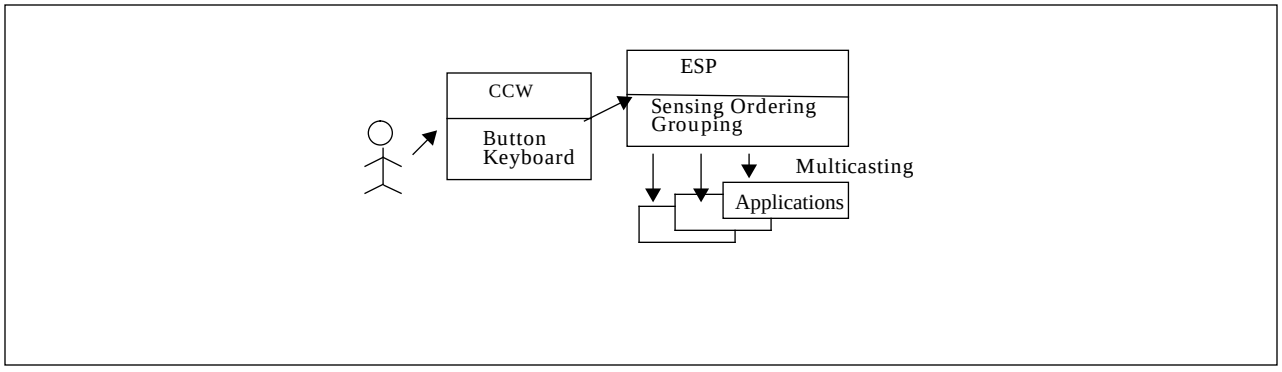
- Built on a multiple client-single server model and uses standard window interface. There are no special libraries, and there is no need to compile or even re-link the existing application source code. This mechanism should be portable to all window systems.
- Provides event sensing and multicast mechanisms to make selected processes execute concurrently.
- Provides a concurrent dialog to a group of applications to execute multiple functions simultaneously.
- Provides layered GUI interface to allow graduated access to the more powerful and complicated GUI capabilities.
- Provides a simple point-and-click approach to dynamically connect or disconnect selected existing applications in a working context.
- Provides dynamic context for multicasting.
- Provides dynamic button assignment for heterogeneous applications.

2.1 ESP Architecture

ESP is built on a multiple client-server model. Figure 1 illustrates the overall architecture. It contains two key components:

1. Concurrency Control Window (CCW)
2. Event Sense and Muticasting Processing (ESP)

FIGURE 1.ESP Architecture



Applications use windows to communicate with users. Users request an application to perform a function by sending events, such as a result of a key, mouse button, or sprite motion. The application GUI behavior understands its window events and how to interact with the application. This mechanism is started by sensing all the possible events happening on an application window.

In general, there are two types of windows commonly used by applications to interact with users: buttons and text fields. The event type received on these windows are button press, button release, key press and mouse movement. Our mechanism senses the common events and puts them into the following two categories: keyboard events and mouse events.

Both keyboard and mouse events contain key press/release and button press/release. Both events are sent by the server to the application when the user presses a key on the keyboard or presses the mouse. Only an application that has specifically asked to be informed of that type of event will receive them.

ESP senses and selects the above standard common window events as the potential candidates to be managed and multicast. However, additional events may need to be included for processing special types of new application windows.

2.2 ESP Operation

ESP operation comprises the following four major steps:

2.2.1 Access Running Application Windows

The purpose of accessing running application windows is to find the window characteristics, their child windows, and their identifiers. This mechanism uses window query tree system calls, window images, or pointer movement to find the corresponding application window structure and its identifiers. Normally, buttons and text fields are the children of the application main window. Buttons and text fields are the input windows used by users to interact with the application.

2.2.2 Build CCW and Child Windows

This mechanism builds the Concurrency Control Window (CCW) to control and manage the activities in the event sensing and multicasting session. In order to receive the same event types, the CCW has to simulate/build the same type of child windows as the application, such as a button to represent the application's button, a command line input field to represent the application text window.

2.2.3 Sense User Window Events

Using a CCW (Concurrency Control Window), this mechanism senses the user actions/window events and passes them to the corresponding application windows to execute some function concurrently. We use Xt Intrinsics multiple input application contexts and registered event handlers in our current X Windows prototype. Similar methods can be applied to other window systems.

2.2.4 Multicast User Events

After building the same type of window, ESP controls and multicasts the window events received from the end user to the selected windows to execute a user command such as simultaneously changing the sales tax rate in all participating spreadsheets. Using this mechanism, the user can concurrently manage the existing application's execution.

There are two modes of operation:

1. Global:

   Using CCW window global buttons to invoke all or a selected group of the running applications to perform a function simultaneously.

2. Local:

   Using CCW's individual selection buttons to raise a specific window for executing an independent function.

Methods for using these modes include the following:

In the global mode, the application GUIs may be structured in a hierarchical fashion with an arbitrary number of levels. Interacting with a GUI leads to window events being generated on one or more of the children GUIs. The mapping of window events to children GUIs is many-to-many. The children applications may be sequential or concurrent. Window events on children GUIs lead similarly to further events being generated on grandchildren GUIs. As an example of multiple events being generated for a single child GUI, a show button on the CCW window could be mapped to both the data visualization and performance visualization buttons on the same child visualization application.

Child GUIs may be specialized to different functions available in different applications or even different components of the same sequential application. In the local mode, ESP allows the independent construction of the different functional GUIs. A user can interact with the CCW window and access some generic functionality or access the individual children GUIs for access to specialized functions. For example, a parallel application could be constructed out of a Lotus 123 spreadsheet application, Pablo [2] (a performance visualization application), and a mail program in order to do some spreadsheet calculations on raw performance data, graph computed visualization data, and mail it to a distribution list. Specialized GUIs are provided by the individual application GUIs whereas generic and composite functionality is provided by the CCW window.

2.3 ESP Operations Extensions

ESP operation includes the following three extensions:

2.3.1 Dynamic Window Association

ESP can associate the generic windows, such as buttons, text fields, with a specific window in the same or different applications. In addition, a group of application buttons can be associated with a generic CCW button. With ESP, users can dynamically associate their CCW buttons with any heterogeneous function they want to perform at run time.

2.3.2 Event Concurrency Control, Grouping, and Synchronization

ESP uses a layered structure to concurrently control the selected events happening in the existing application GUIs. One single user request may generate multiple dialogs with the applications which depend on the structure of the application GUIs.

The user can dynamically add or delete an application window to or from the working context. Each application window is only used for processing local application functions.

ESP allows the user to synchronize different types of window events to manage the order of execution by the applications. When associating the application buttons with a CCW button, the user can indicate the order of execution and which group of functions can execute simultaneously.

2.3.3 Enabling Full-Screen Applications

This mechanism can be also used on the applications that allow the user to interact with the screen anywhere on it. Example interactions include text insertion, button press etc. We make our CCW an exact replica of the application window. Putting the replica directly under one of the instances of the application window and setting the focus to the replica, makes it looks to the user like everything typed in one application window is simultaneously being typed in all instances of the application in the current context.

3.0 Integration of ESP with Collaborative Computing

Human endeavors involve more than one person, especially in a complex distributed and parallel application. Different expertise is need to work on different parts of a program. At times, it is appropriate to examine a single running instance of an application with multiple experts. For example, we can use HP SharedX to allow more than one person to observe and control the program on a remote workstation. However, there is no global control mechanism to allow users to control concurrently the interactions with a set of shared applications.

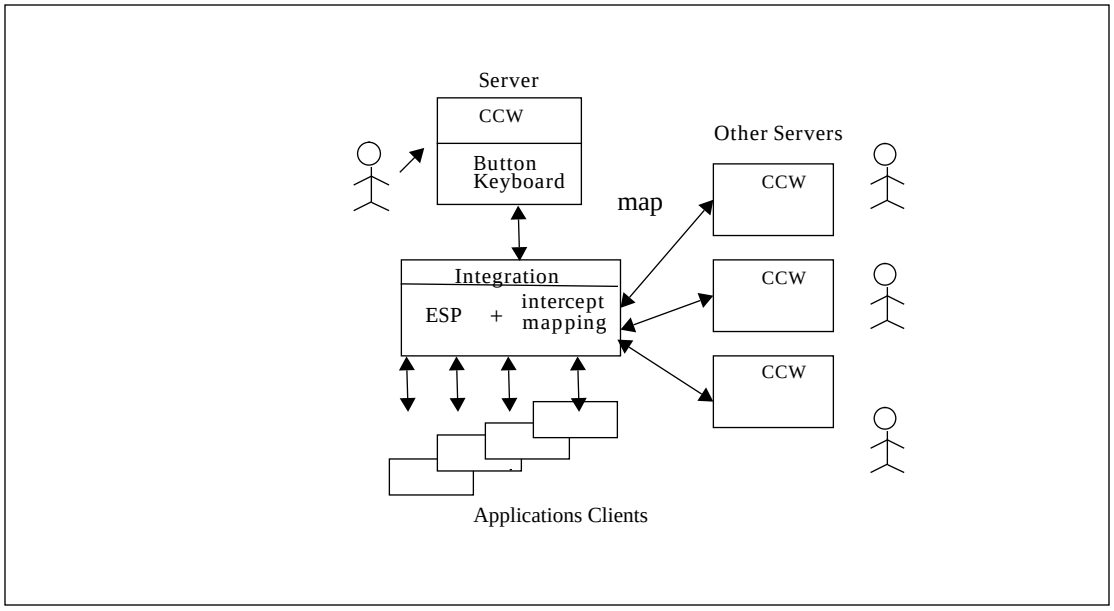
ESP solves the problem of how to sense user actions, to control, and to distribute input window events to each selected application window for simultaneous execution in a distributed environment. The integration of ESP with collaborative computing achieves the goal of sharing multiple applications among users. It contains three key components:

1. Concurrency Control Window (CCW)
2. Event Sense and Muticasting Processing (ESP)

3. Intercept requests from application clients and map them onto the other servers [11]

Figure 2 illustrates the integration of ESP with collaboration.

FIGURE 2.An Integration of ESP and Collaboration



4.0 Two Examples

The examples presented here show how potential applications might use ESP. The first example is IVD [1], a Hewlett-Packard Laboratories Interactive Visualization Debugger working prototype for message passing parallel applications.

The second example is a commercial banking application that coordinates multiple spreadsheets, a plotting program, and a mail program.

4.1 A Parallel Visualization Debugger

PVM (Parallel Virtual Machine) [6] is a popular system for writing parallel applications. PVM requires a debugging interface that will support debugging functions for all selected processes using a single window. Such an interface will be very valuable for actions such as simultaneous single-step execution in all selected instances. At present, PVM uses a separate window for each selected process. Users have to enter commands in each debugging window.

IVD uses ESP to integrate existing debuggers and visualizers to debug/observe parallel application execution behavior in a distributed, shared-nothing multicomputer environment with the following features:

4.1.1 Heterogeneous processing

ESP is designed for heterogenous processing. Using ESP input event sense and multicasting capability enables IVD to access existing debuggers across various platforms to debug large, complex problems.

4.1.2 Scalability by grouping

ESP is scalable. Each process has its own debugging and visualization facilities. ESP provides different "contexts" for the user to dynamically select the multicasting scope. By sending commands only to processes within the single, currently active context rather than all processes simultaneously, the user may limit the amount of overhead generated by running multiple processes.

With a context, the user can group certain debugger instances together. Each context then receives the same debugger commands entered on the IVD concurrency control window. By grouping the debugger instances, the user can indicate the execution order of groups of debugger instances. The user is not restricted to work only with individual debugger instances. This becomes useful when certain debugger instances are naturally associated with one another. For example, one context may contain all the slaves while another context contains only the master. Alternatively, there may be many contexts, each containing the debugger instances that are functionally related to one another. A given debugger may belong to more than one context or to no context.

4.1.3 Dynamic Context Modification

Figure 3 illustrates the mechanism to dynamically group a set of application programs to run simultaneously.

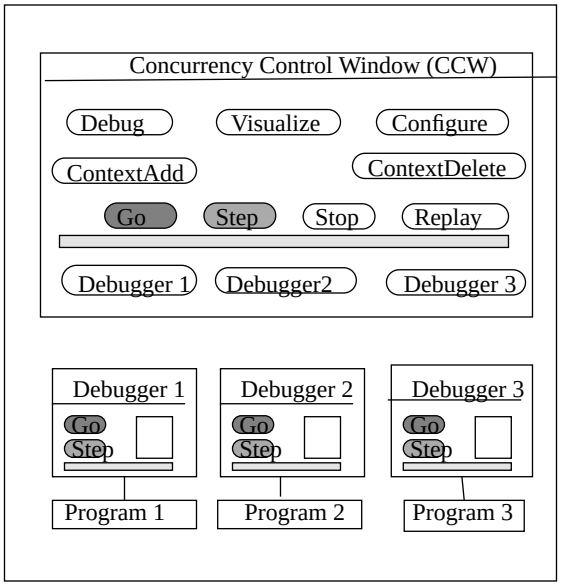
A user starts a visualization and debugging session in a parallel programming environment. The user starts up the session by bring up a Concurrency Control Window. Afterwards, the user performs the following actions:

1. Presses the "Debug" button in CCW to start the debugging environment and bring up the "Debugger 1" window.
2. Types "debug program 1" in the debugger window command line.
3. "Program 1" spawns "program 2" and "program 3" running under "debugger 2" and "debugger 3" respectively on different workstations.
4. ContextAdd debugger1, debugger2, and debugger 3 in the current debugging context.
5. Uses global buttons on the CCW window to issue go, step and other debugger command.
6. Types setting breakpoint, print commands in the multicast command line.
7. Steps through application 1, 2, 3 concurrently to locate the problem.
8. Press global "step" button to simultaneously single-step through the programs 1, 2, 3.
9. ContextDelete "Debugger 2" from the working set. Step through programs 1 and 3.
10. ContextAdd "Debugger 2" to the current working set.

11. Types “print a” to ask programs 1, 2, 3 to print the variable “a” in their windows simultaneously.
12. Press “Go” button to ask program 1, 2, 3 to continue execution simultaneously

Repeat the above global/local debugging operations until the bug is found.

FIGURE 3.Dynamic Context Modification

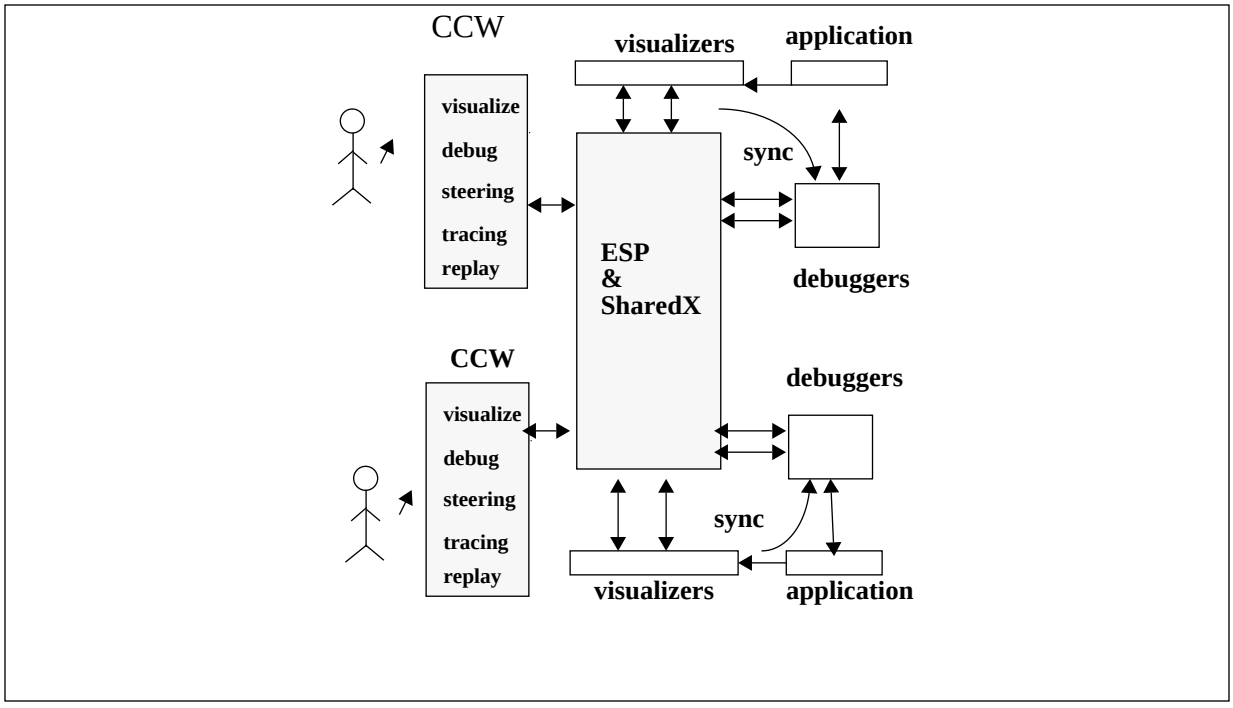


4.1.4 A Multi-User collaborative Debugging Session

Figure 4 shows that IVD with ESP appears as a single visualization debugger to the user. ESP manipulates the user input events, such as keyboard and mouse, and then multicasts these events to each running visualizer and debugger. ESP automatically triggers the visualizers/debuggers to execute received events from the user. User events are processed as if the window events had been directly entered into the visualizer/debugger windows.

HP SharedX allows more that one person to share and control a single running instance of multiple applications including IVD. Hence a systems expert and the program developer can work together to fix the program.

FIGURE 4. Collaborative Debugging

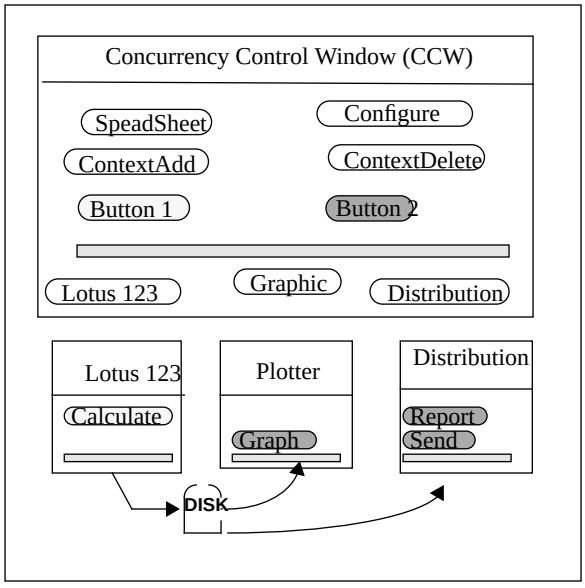


4.2 A Commercial Banking Application

Consider a group of departments that run a spreadsheet each month, plot some graphs, and print a report. The normal mode of operation is to have someone build the spreadsheet templates including formulas, set up the graphics formats, and build the reports. These are distributed to machines in each department. Secretaries in each department enter the department specific data into the work space and run the graphs and reports. Any changes to the spreadsheet formulas or report formats must be distributed to each department and updates made locally in a timely manner.

With our mechanism the procedures would be somewhat simpler. The person responsible for maintaining the spreadsheet would bring up a control window. The spreadsheet application would be started on each node using the facilities of the control window, and a flag would be set to indicate that changes made to one machine's copy would be made to all. Next, typing in one window, the user would build the spreadsheet, graphics formats, and reports. Changes would be made the same way. The department secretaries still enter the department specific data. At the end of each month, one person could bring up the spreadsheets on all the machines, set the context so that commands are globally executed, and generate all the graphs and reports by typing the commands in only one window.

FIGURE 5.A Banking Application Association



An extension to this example is to use ESP to control completely different applications. The bank uses spreadsheets to calculate its annual result, then uses the specific graphic and distribution programs to plot, produce reports, and send to all the customer. The generic “Button 1” on the CCW associates with Lotus 123 “calculate” button. The generic “Button 2” associates with three buttons: “Graph” on the graphic program window, “Report” and “Send” buttons on the distribution program windows.

It only takes the accountant two steps to distribute the annual reports:

- Press the “Button 1” to compute the annual income and expense. The results are stored on a disk.
- Press the “Button 2” button to read data from the disk, make the graphic charts, produce the annual reports, and distribute the reports to the customers. All three steps are performed by pressing the “distribute” button and all three functions are executed concurrently. ESP allows the user to define the ordering of the event to be sent.

5.0 Related Work

Research in the area of asynchronous collaborative computing has been published under a variety of subjects in the recent past.

The Lotus Notes [8] product allows users to get instant access to the same information without any further intervention by sender or the receiver. This is similar to our commercial banking example using multiple, shared spreadsheets. There are two differences, however. First, ESP allows multiple users to manipulate multiple applications directly and not just update data files.

Second, Lotus Notes uses a relational database whereas ESP multicasts uninterpreted window events directly to application windows.

A couple of other systems actually allow concurrent manipulation of multiple applications by multiple users. A system from IBM[9] uses global cursor for concurrent data entry and manipulation in multiple applications. A system from Wang Lab [10] routes keystrokes to processes which are associated by reference to a a routing table. However, most of these concurrent control of multiple applications have limitations. IBM and Wang's mechanisms require cursor position tracking and keystroke interpretation while inputting common data into a plurality of programs. The overhead cost is very high in both. ESP provides a concurrency control window to receive, order, group and manage incoming window events and to forward them to running applications. ESP does not interpret keystrokes and pointer movement. Further, this mechanism can access existing application GUIs at run time without changing source code. There is no recompilation, or re-linking, and no special library is needed

ESP can be used in many situations. We have already shown that it is the foundation on which to build an interactive, parallel, visualization debugger in a distributed and parallel environment. We have also noted that this event sense, control, and multicast ESP allows us to mix different application GUIs, such as to combine four applications' buttons — calculate, graph, report, and send — to generate a report and mail it. In addition to being used in a parallel debugging session, this ESP is directly applicable in the following applications:

1. Multiple database queries and updates
2. Networking and manufacturing control
3. Multiple shadow file creation and updates
4. Multiple updates for spreadsheet and banking applications

In general, ESP applies to all GUI applications. It allows us to take any application GUI and to make it distributed. It allows the end user to control and manage multiple dialogs with existing applications without impact on the existing system or application software. In addition, with SharedX, ESP makes collaborative computing possible to control existing applications concurrently in a distributed heterogeneous environment. ESP provides a singe image in a multi-user, multi-application distributed environment.

6.0 Summary

ESP provides unique features not found in other collaborative computing systems as shown in the previous sections. IVD uses ESP to provide an on-line interactive visualization debugging tool. In addition, ESP has the ability to group processes into various contexts and perform simultaneous operations. Programmers can use their favorite tools. The most interesting feature is that many different, unmodified tools can be accessed simultaneously by multiple users in a distributed

shared-nothing environment. With ESP enabling technology as a vehicle, we will continue to explore collaborative computing with multiple applications.

7.0 Acknowledgment

Thanks to Dr. Chris Hsiung and Dr. Mehdi Jazayeri for their encouragement and suggestions, and to Charles Young and Don Carfinkel for their technical discussions.

8.0 References

[1] Ming C. Hao, Alan H. Karp, Milon Mackey, Vineet Singh, Jane Chien "On-the-Fly Visualization and Debugging of Parallel Programs", Proceeding of the IEEE/ACM MASCOTS 94, January 31, 1994, Durham, North Carolina.

[2] Pablo Instrumentation Environment, Department of Computer Science, University of Illinois, Urbana, Illinois, 1992

[3] John R. Portherfield "Mixed Blessings" and HP "SharedX" HP Professional Volume 5, Issue 9, HP SharedX Unix Today, May 27, 1991.

[4] Terrence Crowley and Harry Forsdick, "MMConf: The Diamond Multimedia Conferencing System", BBN System and Technologies, INC. Aug 23, 1989

[5] BBN/Slate, BBN Software Products, Cambridge, M.A. 1990

[6] G. A. Geist, "Network Based Concurrent Computing on the PVM System." Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 3781. 1992.

[7] Communications of the ACM Special Section on "Graphical User Interfaces: The Next Generation", April 1993, Volume 36, Number 4

[8] Lotus Notes Users' Guide, Lotus Development Corporation, Cambridge, MA, 1989.

[9] Robert J. Torres, "Method for concurrent data entry and manipulation in multiple application", IBM patent application, September, 1989.

[10] Deborah A. Rhodes, Waltham Mass, Eric Rustici, Kelly H. Carter, "Method for routing events from key strokes in a multi-processing computer system", January, 1992, Wang Laboratories, Inc.,

[11] M. C. Hao, F. D. Snow, J. Chien, J. A. Latone, "Experiments in Collaboration with Existing X Applications", G32-3569, May, 1992, IBM Palo Alto Scientific Center.

HEWLETT PACKARD

Concurrent Application Control In Collaborative Computing

Ming C. Hao, Alan H. Karp, Vineet Singh

Jane Chien

HPL–94-37

Event sensing,

Concurrent control,

Multicasting window events

Distributed collaborative

Heterogenous

Most people think that collaboration implies that several people are sharing work on a single application with shared displays. In fact, collaboration is more. It includes the concurrent control of multiple applications by a collaborative group. To enable this more powerful form of collaboration, we show how to combine earlier mechanisms for single client, multiple server computing with a new mechanism called ESP (Event Sense Protocol) for multiple client, single server computing. We describe two extended examples — a working prototype of a multi-user, heterogeneous, parallel debugger and a commercial banking application.