Collaborative Design Using Your Favorite 3D Application

Ming C. Hao, Joseph S. Sventek
Software Technology Department
HPL-96-51
April, 1996

concurrent engineering, even-driver, collaboration, application sharing

We describe an event-driven application sharing (EDAS) technology which enables the collaborative design of artifacts using existing, 3D, graphical design applications. The basis of the technology is the capture, multicast, and synchronization of window system events among multiple copies of an application executing on different workstations. This capability is achieved without modification to the applications or to the window system software.

Collaborative Design Using Your Favorite 3D Applications

Ming C. Hao and Joseph S. Sventek

Hewlett-Packard Research Labs, Palo Alto, CA

Abstract

We describe an event-driven application sharing (EDAS) technology which enables the collaborative design of artifacts using existing, 3D, graphical design applications. The basis of the technology is the capture, multicast, and synchronization of window system events among multiple copies of an application executing on different workstations. This capability is achieved without modification to the applications or to the window system software.
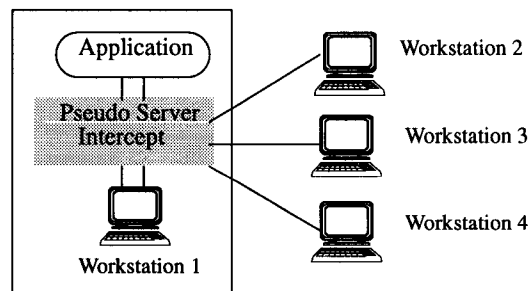
1.0 Introduction

The traditional serial development of a product has been found to result in difficulties, such as long development time and designs that are difficult to manufacture. Industries, like automobile and mechanical manufacturing, recognize the importance of concurrent team design. They are working towards new direct engineering and virtual co-location strategies. They want their designers and manufacturing engineers to work together on large complex design problems.

However, it is not always possible to bring geographically separated groups of designers and engineers together. Attempts to use existing concurrent engineering tools between geographically-dispersed engineers have encountered serious latency problems due to insufficient bandwidth to support the sharing of graphical output from CAD/CAM tools. Users at separate workstations need to be able to share the contents of the windows and communicate with their colleagues independent of the available network bandwidth.

There are numerous collaborating systems that have been designed to provide real-time collaboration. One method uses a network to distribute the graphic output of an application to different displays. Another method uses a pseudo-server to intercept and multiplex window messages from a server to applications. However, most of them usually result in heavy traffic load or demand for high network bandwidth communication links.

FIGURE 1. An Interception scheme

As illustrated in Figure 1, a common method is to share a window among multiple workstations for the same application (client), then to use a pseudo server to intercept existing application messages and pass them to each user's workstation. To the application, the

pseudo server looks like the real server. To each user's workstation, the pseudo server looks like the client. The data exchanged between the user application and the pseudo server is identical to the data exchanged between the pseudo server and the user workstations. It usually generates heavy network traffic due to continual shipping of graphics primitives and bitmaps among the participating workstations.

JVTOS [1, 2] uses a pseudo server to simultaneously share single-user applications on the assumption of a high-speed network. SharedX [3] extends the X window system to allow application sharing by replicating the X protocol stream for each of the target windows. Since the output protocol stream is relatively high bandwidth, this scheme does not work well for large numbers of targets or for high bandwidth applications.

Another drawback to the pseudo server scheme is that the user can only share the graphics calls directed to the pseudo server. Most 3D X Window applications use direct hardware access (DHA) to display sophisticated graphics. DHA allows applications to bypass the X server to render graphics on the displays.

To date, most application software needs to be modified to support collaboration. Rohall's [4] CSCW infrastructure is designed to handle heavy-weight data sets, such as large image and video with good response time. However, it requires modification to applications to use special communication protocols. MMConf [5] requires re-linkedit of the user interface toolkit into the application before sharing.

Our goal of the concurrent engineering collaboration work at the Hewlett-Packard

Research Laboratory is to solve these problems, namely:

- work well over a typical spectrum of network bandwidths
- require no change to existing application and window systems
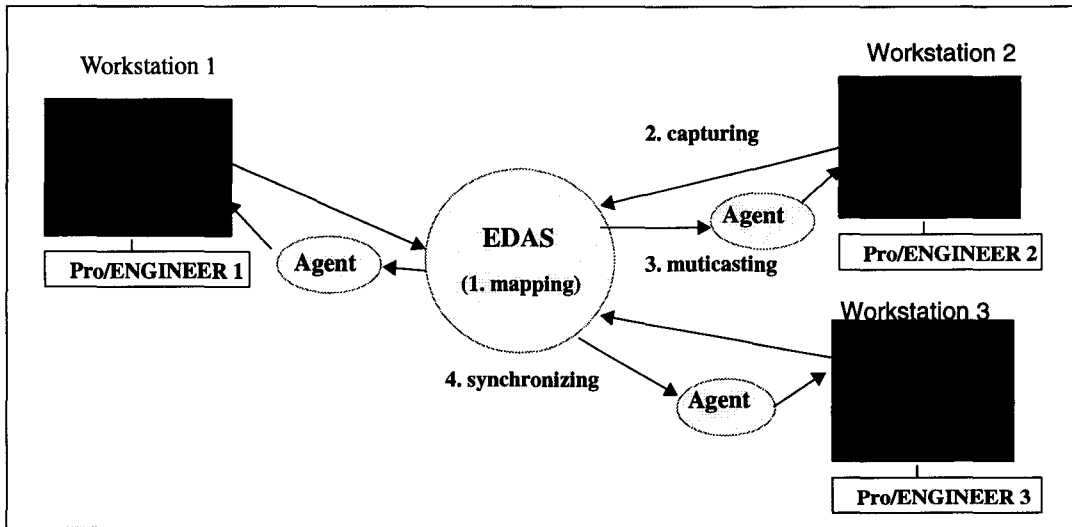- support real-time 3D rendering

2.0 Our Approach

In a non-collaborative setting, once a user has started a 3D, graphical design application, all interactions between the user and the application occur through the delivery of events by the window system to the application; examples of user actions which cause events to be delivered include button press, button release, key press, key release, pointer motion, window resizing, and window creation.

Rather than use a pseudo server scheme to intercept window messages, our technology captures such user-generated events targeted at an application configured for collaborative use. After the capture, they are processed and multicast to all of the applications running on the different workstations.

This approach assumes that if an identical set of applications are started in the same initial state and process an identical sequence of events, then the applications will continue to be in an identical state after processing each event.

2

FIGURE 2. A new approach

Consider the example illustrated in Figure 2. EDAS allows engineers/designers using ProEngineer, a popular CAD application, to concurrently design a mechanical bolt in real-time from different workstations. Unlike a pseudo-server which replicates all graphical output among the applications, this mechanism replicates only the input events among applications. Our mechanism provides application sharing through multiple, synchronized instances of shared applications. The current EDAS includes: (1) event mapping: to map the corresponding window tree structure of each application; (2) event capture: to capture user interactions on the window, such as mouse motion, a button press, or a key press; (3) event multicasting: to multicast input events from a single application window to some group of existing applications running on different workstations; (4) event synchronization: to maintain a consistent view among the collaborating applications.
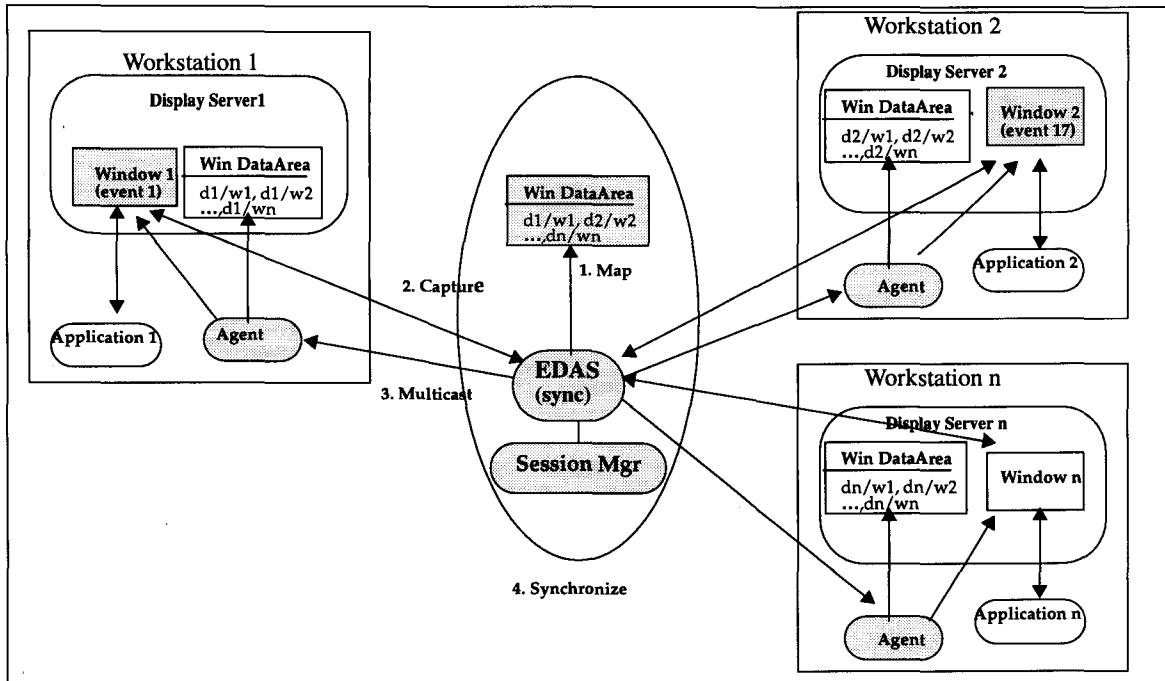
3.0 The EDAS Architecture

In this section we describe the application sharing design based on EDAS. EDAS is a mechanism which is built on a multiple client-server model. Figure 3 illustrates the overall architecture. It contains the four basic components:

1. Event mapping
2. Event capture
3. Event multicasting
4. Event synchronization

These components have the following characteristics:

- built on the window hierarchy structure of the applications
- forces one single input source for easy synchronization through follow control
- separates application program data, execution, and control

3

FIGURE 3. Architecture OverView

This mechanism separates private input events from the shared ones. To achieve fast response time, EDAS replicates application program data and execution on each site. For easy synchronization, EDAS uses a floor control mechanism to guarantee that a deterministic event stream is sent to the shared applications for processing

After input events are captured, EDAS analyzes the events, puts them in proper execution order, and multicasts them to each shared application. As needed, EDAS slows down the event processing to synchronize multiple views. EDAS employs an agent on each remote workstation to: (1) retrieve and send the local application window hierarchy array for mapping; (2) maintain event consistency among shared applications; (3) multicast incoming events; (4) mediate environmental differences to allow for different window sizes, key codes, colormaps, etc.

In Figure 3, the EDAS's main functions include: (1) mapping the local shared application window hierarchy data arrays from workstations 1-n to a global window hierarchy data array; (2) allowing users to enter the shared application window; (3) capturing user's window input events, such as event 1 at window 1, event 17 at window 2...; (4) mediating the incoming events as needed; (5) synchronizing motion events as needed; (6) multicasting the events to windows 1-n to execute some functions.

Each of these components is described further in the following sections.

3.1 Event Mapping

By using the window hierarchy structures and not window attributes, we incorporate a means to automatically map a window in one site to a window in other sites under the same window hierarchy. EDAS communicates with each agent residing at different workstations to construct various instances of the 1-1 parent-child window mapping.

The window system automatically invokes each corresponding application to execute the incoming input events from EDAS. In some situations, we may wish to do the mapping interactively. In this case, the user of the application may bring up his or her windows and invoke an EDAS Map-Window function. The user would use mouse clicks to connect together those shared windows. Once this mapping exists, it can be processed by EDAS.

3.2 Event Capturing

Today, as soon as the user types a character or presses a button on a window, the window event is delivered by the display(server) to the appropriate application (client), the owner of the window [6, 10]. Input events are private to the application. All events occurring on a window will be sent directly to the application.

To share existing applications, EDAS explores a new event capturing mechanism [7] instead of interception. The input events are captured when the user enters the mouse into the shared window. Events from the mouse, key, and button will be sent to EDAS rather then the normal client who would have received them. EDAS processes the events and then multicasts them to the shared application windows to execute some function concurrently.
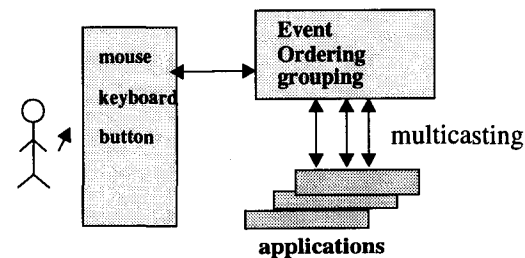
3.3 Event Multicasting

After analyzing the captured input events, the EDAS [7] orders or groups them if necessary and sends the shared input events to the target application windows. Applications automatically trigger their own event handlers to execute received events. Events are processed just as they would be if the window events had been directly entered into the application windows.

The key functions are:

- Grouping: to allow users to select the input event distribution scope.
- Ordering: to sequence events from multiple sources into a proper execution order.
- Multicasting: to distribute events to the appropriate targets.

Figure 4 illustrates the EDAS basic muticasting function flow.

FIGURE 4. Multicast user events

In the sharing of multiple applications simultaneously, EDAS provides different "contexts" for the user to dynamically select the multicasting scope. This becomes useful when certain application instances are not allowed to execute concurrently by all shared applications. For example, the "save" function is not allowed to be executed by all shared applications in a central repository environ-

5

ment. Alternatively, there may be many contexts, each containing the application instances that are functionally related to one another. A given application may belong to more than one context or no context.

3.4 Event concurrency control and synchronization

With a replicated execution approach, EDAS synchronizes multiple copies of a shared application running on each site. The need for synchronization arises due to the mismatch of processing speeds of various processors. Our synchronization mechanism employs motion event compression and a two-phase protocol. EDAS ensures that all shared applications are in a consist state before processing the next incoming event. In case of out of synchronization, EDAS allows users to easily resynchronize the application in local mode.

4.0 Session Management

To start a concurrent engineering design session on a running application, only the collaboration initiator requires EDAS, all other participants' workstations need to run an agent and be connected by a network.

We provide a concurrency control window (ccw) on each participant's screen. The ccw provides an OSF/Motif look and feel user interface which controls the application sharing.
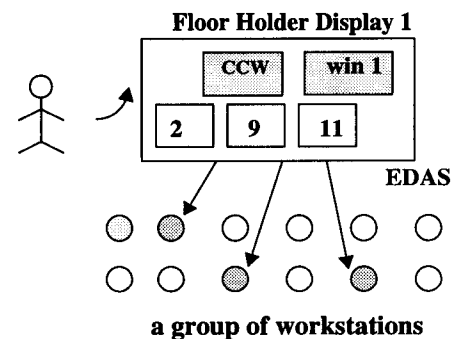
A participant needs to acquire the floor before sharing. It identifies the participant currently providing input events to the session. Input events from other participants are inhibited. Only one participant may have the floor at a time. Participants can take turns with the

"token", which allows each participant to manipulate the 3D model as needed.

We also provide a shared cursor to enhance the collaboration. With a shared cursor, the movement of the floor holder's cursor are visible to other participants.

Figure 5 illustrates the display of the floor holder at workstation 1. There are 12 copies of a shared CAD/CAM application running on 12 workstations. From the ccw window, the floor holder can dynamically select a subset of copies of the application to concurrently execute some functions. For example, during an engineering design session, the floor holder selects applications 2, 6, 11 to construct a bolt. Afterwards, the floor holder selects applications 7, 10, 12 to design a brake. All participating engineers within the selected context can view and change the rendering as needed in each design and review process.

FIGURE 5. A concurrent design session

a group of workstations

5.0 Examples

EDAS is an on-going experiment in Hewlett-Packard Research Laboratories. We built a series of increasingly powerful prototypes to demonstrate and evaluate the key technology in the X Window system. Our
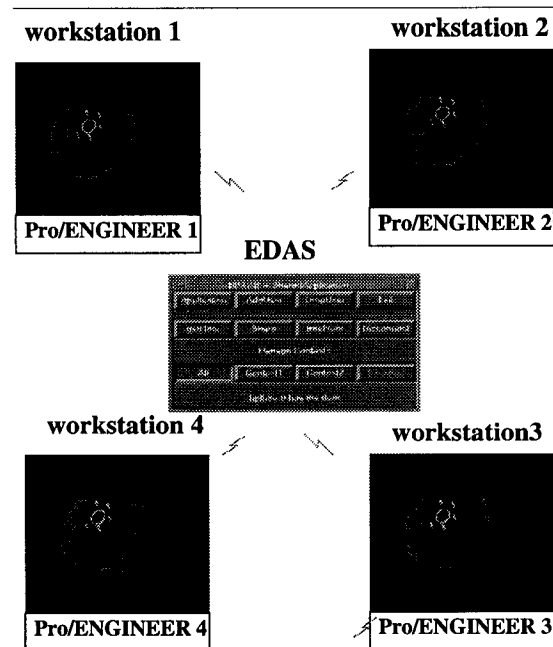
experience with the sharing of unmodified 3D CAD/CAM applications has been promising. The response time is almost instantaneous.

Here is a partial list of the applications that have used this technology for sharing among different X Window workstations.
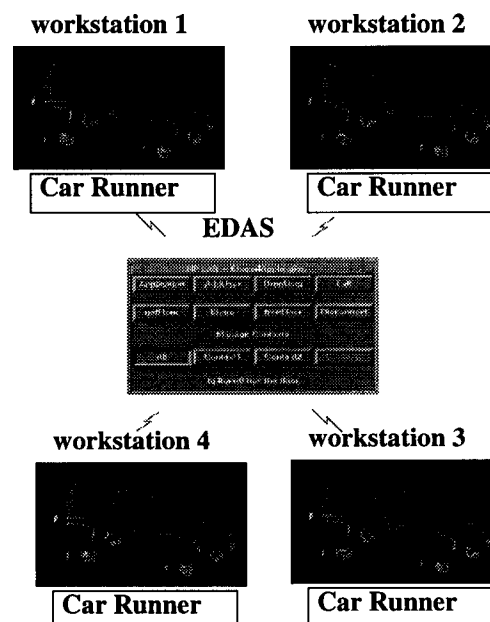
- ProEngineer

  sharing of 3D modeling/rotation/assembly

- ICAD

  sharing of high data volume CAD applications

- PDGS

  sharing of mechanical parts engineering

- World-Wide-Web (Netscape Browser)

  sharing of HTML (Hypertext Markup Language) and 3D graphics

- Runner

  sharing of 3D interactive graphics (Starbase, through DHA)

- Video Player

  share of real-time video presentation (MPEG files)

Figure 6 and Figure 7 show examples of 4 engineers concurrently designing a brake and an automobile together using the Pro/Engineer and Car Runner at different sites.

FIGURE 6. Concurrent Rotating

FIGURE 7. Sharing 3D rendering

7

6.0 Conclusions

EDAS employs new underlying design mechanisms not found in current 3D concurrent engineering products. Most software collaboration research prototypes and products support file/image/store-forward sharing.

EDAS has been used to support collaborative CAD/CAM 3D modeling among multiple workstations. Its event-multicast design center permits it to be usable over a variety of network bandwidths (tested as low as 56 kbps).

In summary, EDAS meets the criteria on application sharing architectures[8, 9]. With the separation of program execution and control, EDAS achieves the fast response time through replication. With the direct sharing of user interactions, EDAS resolves sharing of 3D rendering issues. With a single user input source and two-phase protocol, EDAS maintains a consistent view among shared applications. In addition, EDAS provides high security without intensive encryption since only events are shipped, not the data. EDAS is a simple and significant technology to support real-time, 3D concurrent engineering.

Acknowledgment

References

[1] Thomas Gutekunst, Daniel Bauer, et al, "A Distributed and Policy-Free General Purpose Shared Window System", IEEE/ACM Transactions on Networking Vol3, No. 1, Feb. 1995.

[2] Thomas Gutekunst and Bernhard Plattner, "Sharing Multimedia Applications Among Heterogeneous Workstations", Second International Conference on Boadband Islands, 1993 Elsevier Science Publishers B. V.

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

[4] Stephen Zabele, Steven L. Rohall "High Performance Infrastructure for Visually-intensive CSCW Applications" CSCW 94, Chapel Hill, NC.

[5]Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick, and Raymond Tomlinson, "MMConf: An Infrastructure for Building Shared Multimedia Applications" CSCW 90, Los Angeles.

[6] Adrian Nye, "Xlib Programming Manual" O'Reilly & Associates, Inc. July, 1992.

[7] U. S. Patent pending on "A Mechanism to Control and Use Window Events Among Applications in Concurrent Computing" and "A Mechanism to Sense and Multicast to a Plurality of Exisiting Applications for Concurrent Execution in a Distributed Environment" Hewlett-Packard, Oct, 1994 and May, 1993.

[8] M. Sobolewski, et al, "Functional Specifications for Collaboration Services", Proceedings of 3rd IEEE Workshop on Enabling Technologies: Infrastructure of Collaborative Enterprises, Apr. 1994.

[9] Daniel Garfinkel, Randy Branson, "A Comparison of Application Sharing Architectures in the X Environment" Xhibition 91.

[10] Communications of ACM, Special Section on "Graphical User Interfaces: The Next Generation", Apr. 1993, Vol. 36, No. 4.